Final Report: Introduction to Devops Terraform Docker Kubernetes

Introduction

This project showcases the development of a sophisticated real-time e-commerce analytics system using AWS services. Leveraging the power of Amazon SQS for asynchronous message queuing and DynamoDB for scalable data storage, this solution enables businesses to capture, process, and analyze e-commerce data in real-time.

Through this analytics system, e-commerce businesses can monitor key metrics such as website traffic, product sales, and inventory levels in real-time. By gaining instant insights into customer preferences, market trends, and inventory status, businesses can make informed decisions to optimize marketing strategies, streamline operations, and enhance the overall customer experience.

Technologies Used in this project:

1. **Amazon Web Services (AWS):**
    - **Amazon SQS (Simple Queue Service):** Asynchronous message queuing service used to decouple the components of the application. Messages are sent to SQS queues by producers and retrieved by consumers, enabling reliable communication between different parts of the system.
    - **DynamoDB:** Fully managed NoSQL database service provided by AWS. DynamoDB offers seamless scalability, high performance, and low latency for storing and retrieving data. In this project, DynamoDB is used for persisting data received from the SQS queue.

- **AWS Lambda:** Serverless compute service used to execute code in response to events. In this project, Lambda functions can be triggered by messages arriving in the SQS queue, allowing for seamless data processing and integration with other AWS services.
- **IAM Roles (Identity and Access Management):** IAM roles are used to define permissions and access control policies for AWS resources. IAM roles ensure that the Lambda functions have the necessary permissions to interact with other AWS services, such as SQS and DynamoDB, while following the principle of least privilege.

2. **Python:**
    - Python programming language is used for writing the backend logic and interacting with AWS services. Python's boto3 library is particularly useful for programmatically accessing and managing AWS resources.
3. **JSON (JavaScript Object Notation):**
    - Lightweight data interchange format used for serializing and transmitting data between components of the application.
4. **IDE (Integrated Development Environment):**
    - Any IDE or text editor can be used for writing and executing Python code, such as VS Code, PyCharm, or Jupyter Notebook.
5. **AWS Access Key and Secret Access Key:**
    - Authentication credentials required for accessing AWS services programmatically.
6. **boto3 Library:**
    - Python SDK for AWS, used for interacting with AWS services from Python code.
7. **Web Browser:**
    - To access and interact with the Streamlit web application or any other web-based interfaces for managing AWS services.

Task 1: Launch EC2 Instance and install Docker

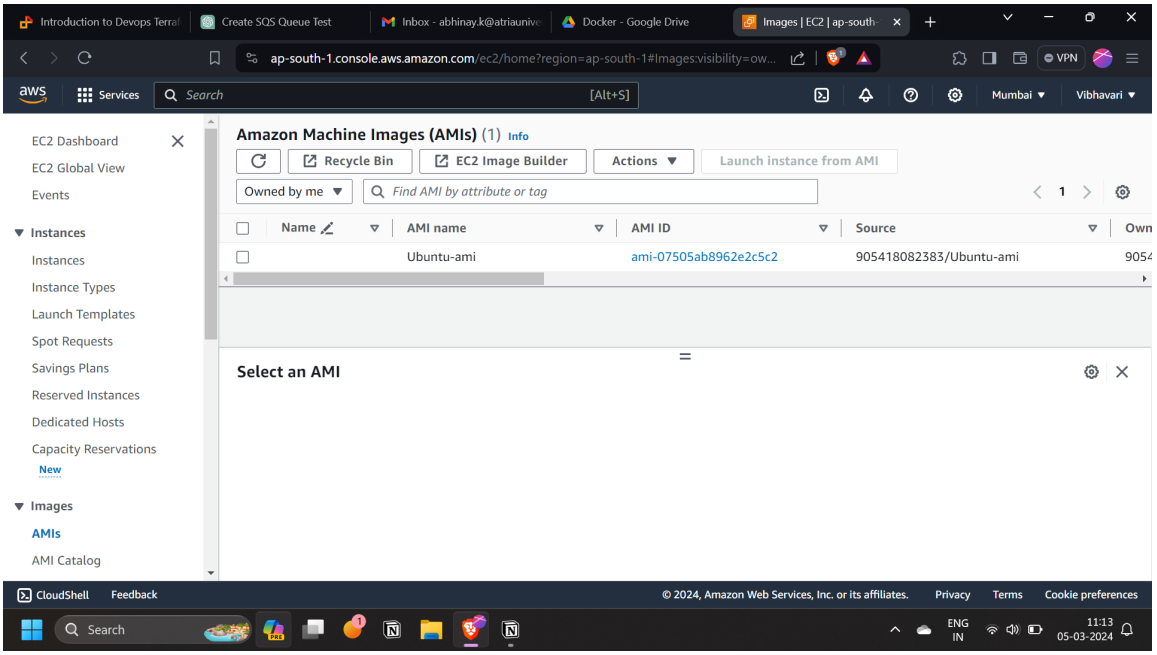
Provisioning an Ubuntu EC2 Instance with Docker Installed using AWS CloudFormation

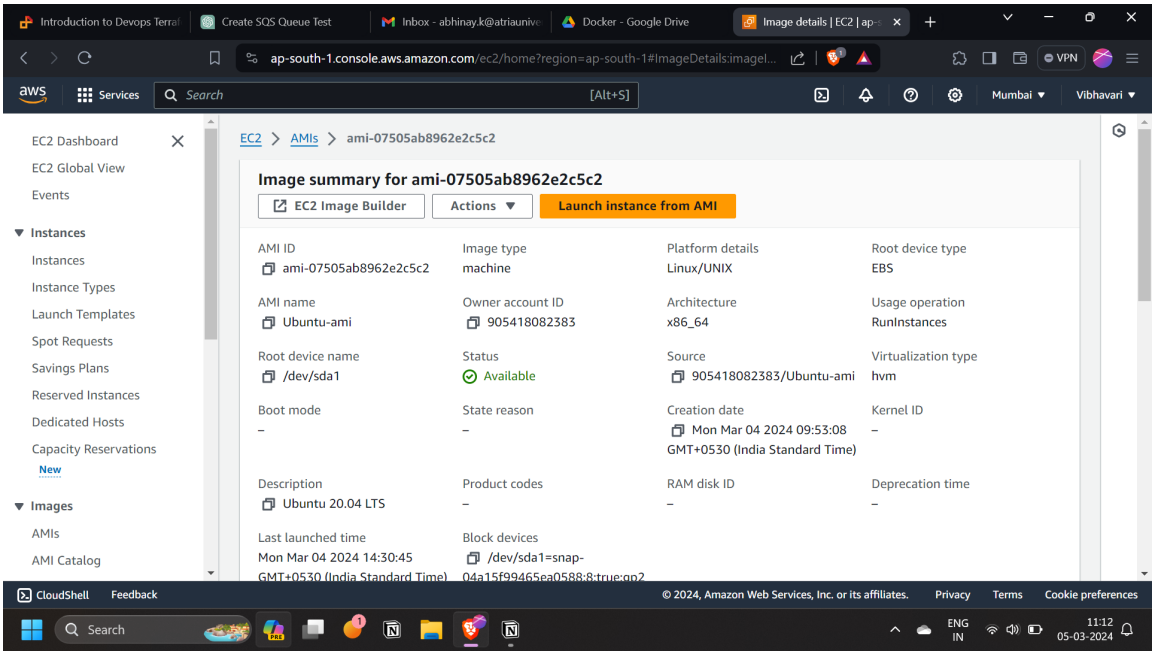
Introduction:

In response to the requirement of TELEMAX, an airline management company, to deploy an effective architecture for storing real-time data in a NoSQL-based database on AWS cloud, this report outlines the steps to create an Ubuntu EC2 instance with Docker installed using AWS CloudFormation.

Step 1: Creating an AMI Template

- Launch an EC2 instance from an existing AMI (Amazon Machine Image) or create a new instance with the desired configuration.
- Customize the instance according to your requirements by installing software, configuring settings, and making any necessary changes.
- Create an AMI from the instance using the AWS Management Console, CLI, or SDKs. This AMI will serve as the template for creating new instances with the same configuration and setup.





Step 2: Creating a CloudFormation Template

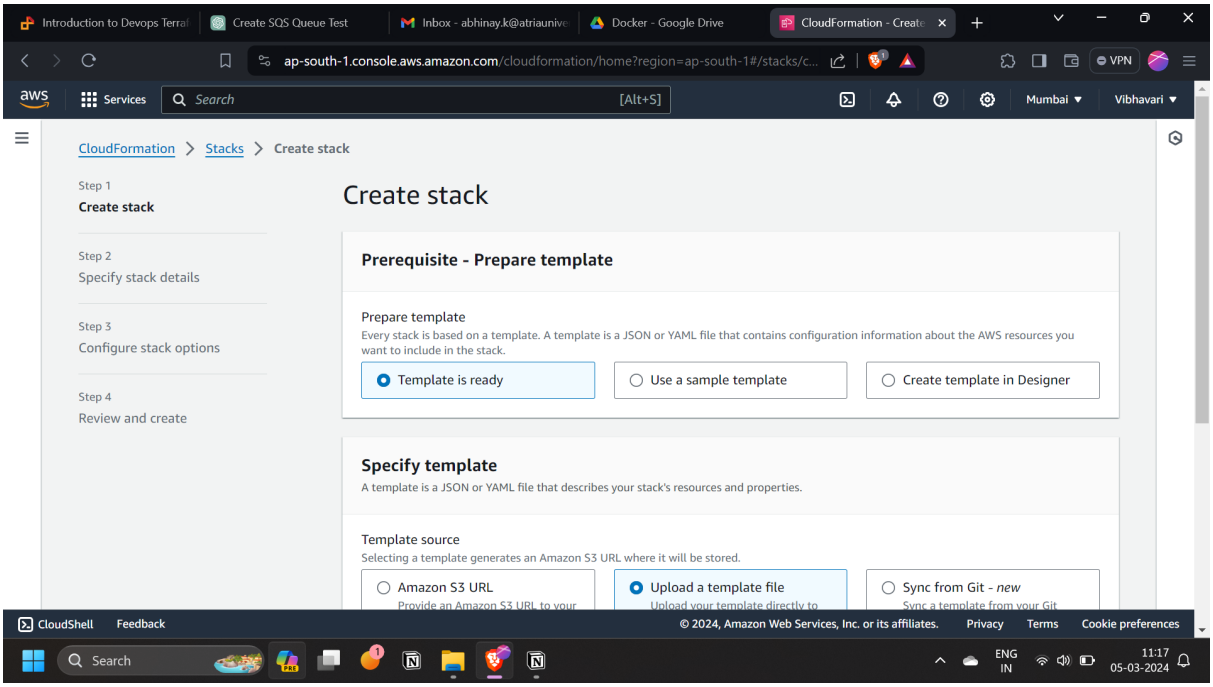
- Define the resources required for your infrastructure in a CloudFormation template. This includes EC2 instances, security groups, IAM roles, etc.
- Specify the properties and configurations for each resource, such as instance type, key pair, security group rules, etc.
- Deploy the CloudFormation stack using the AWS Management Console, CLI, or SDKs, which will create the specified resources based on the template.

Template

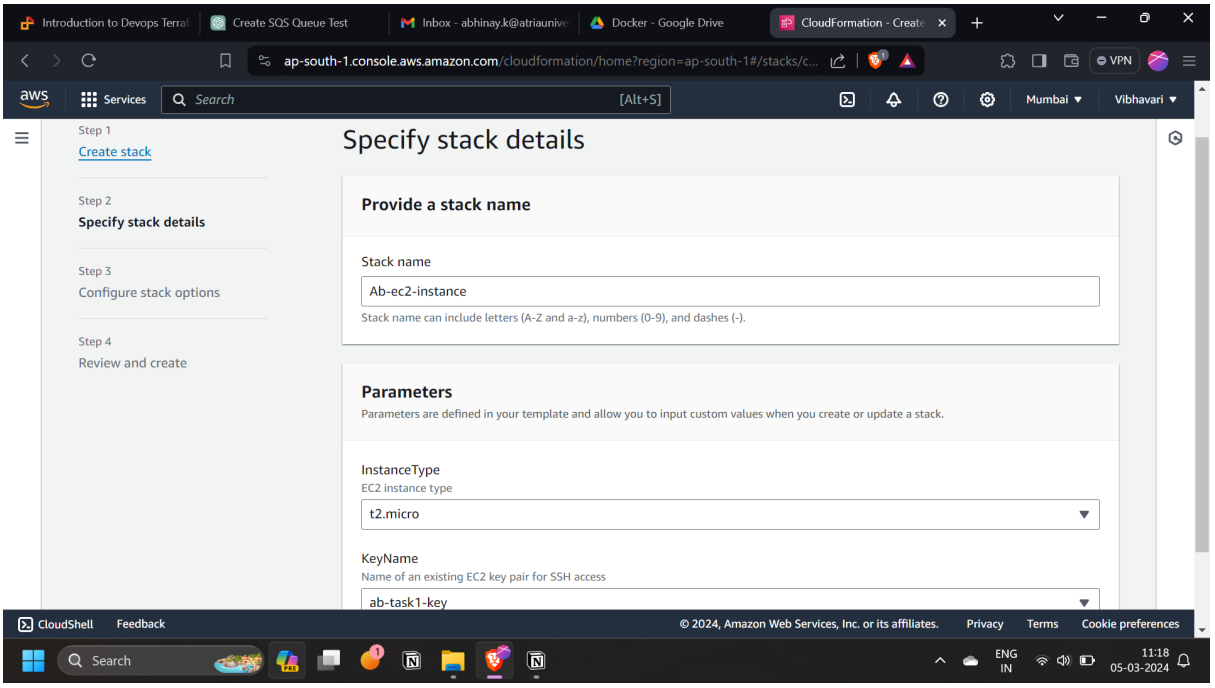
```
AWSTemplateFormatVersion: '2010-09-09'
Description: Create an Ubuntu EC2 instance with Docker instal

Parameters:
  InstanceType:
    Description: EC2 instance type
    Type: String
    Default: t2.micro
    AllowedValues:
      - t2.micro
      - t2.small
      - t2.medium
      - t3.micro
      - t3.small
      - t3.medium
    ConstraintDescription: Must be a valid EC2 instance type.

  KeyName:
    Description: Name of an existing EC2 key pair for SSH acc
    Type: AWS::EC2::KeyPair::KeyName

Resources:
  EC2SecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Enable SSH access to the EC2 instance
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: 0.0.0.0/0  # Allow SSH access from anywhere

  EC2Instance:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: ami-07505ab8962e2c5c2  # Ubuntu 20.04 LTS (HVM
```

```
      InstanceType: !Ref InstanceType
      KeyName: ab-task1-key
      SecurityGroups:
        - Ref: EC2SecurityGroup
      UserData:
        Fn::Base64: |
          #!/bin/bash
          apt-get update -y
          apt-get install -y docker.io

Outputs:
  InstanceId:
    Description: Instance ID of the newly created EC2 instanc
    Value: !Ref EC2Instance
    Export:
      Name: EC2InstanceId
```

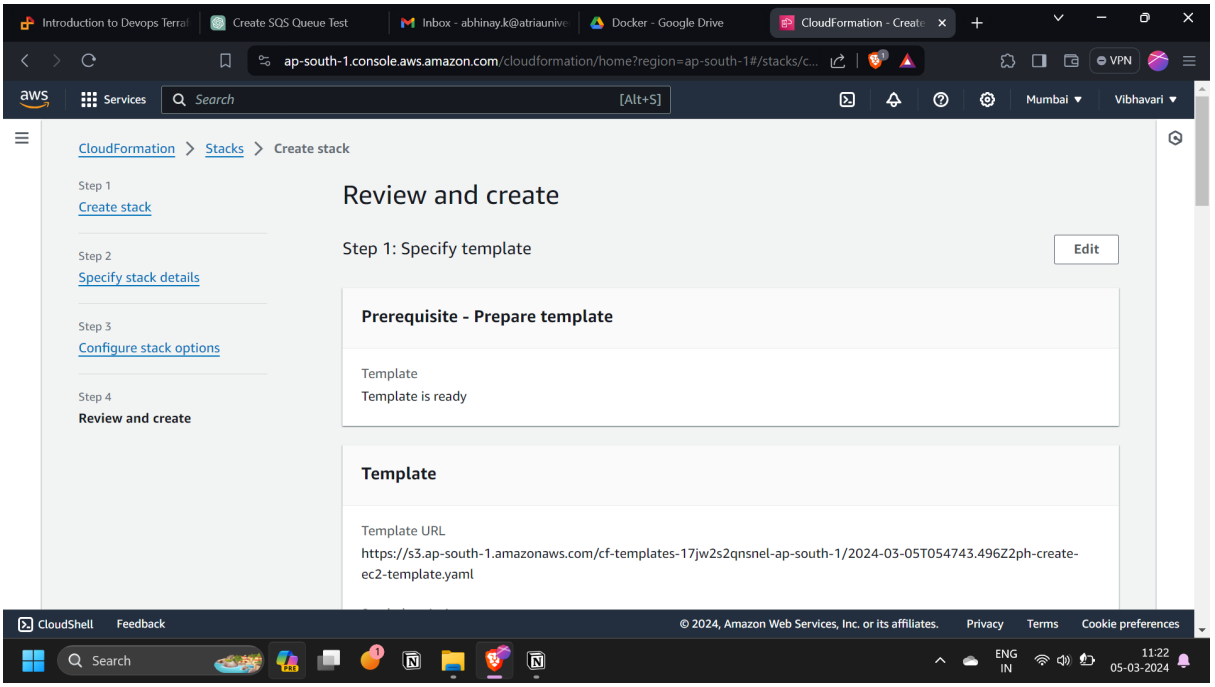
Step 3: Uploading CloudFormation Template:
Click on the "Create stack" button and choose the option "With new resources (standard)". Select "Upload a template file" and upload the CloudFormation template file provided.



Step 4: Configuring Stack Parameters:
Enter a stack name and specify any required parameters such as the key pair name. Proceed to the next steps, confirming the default settings or making adjustments as necessary.

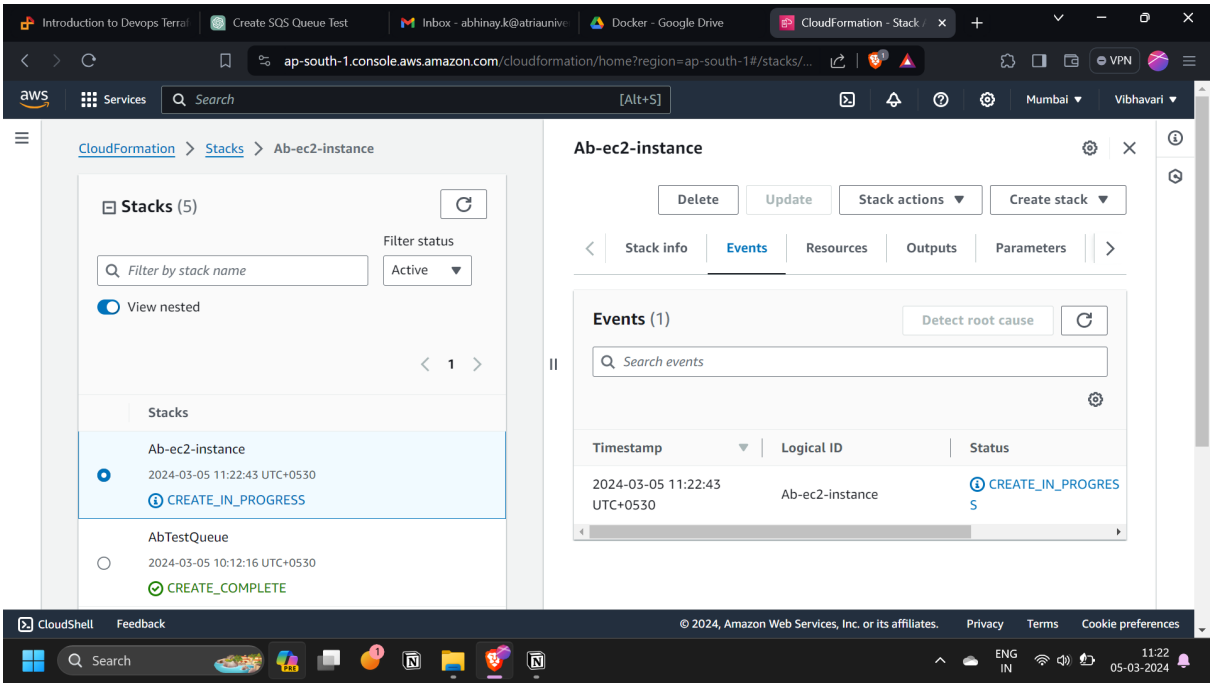


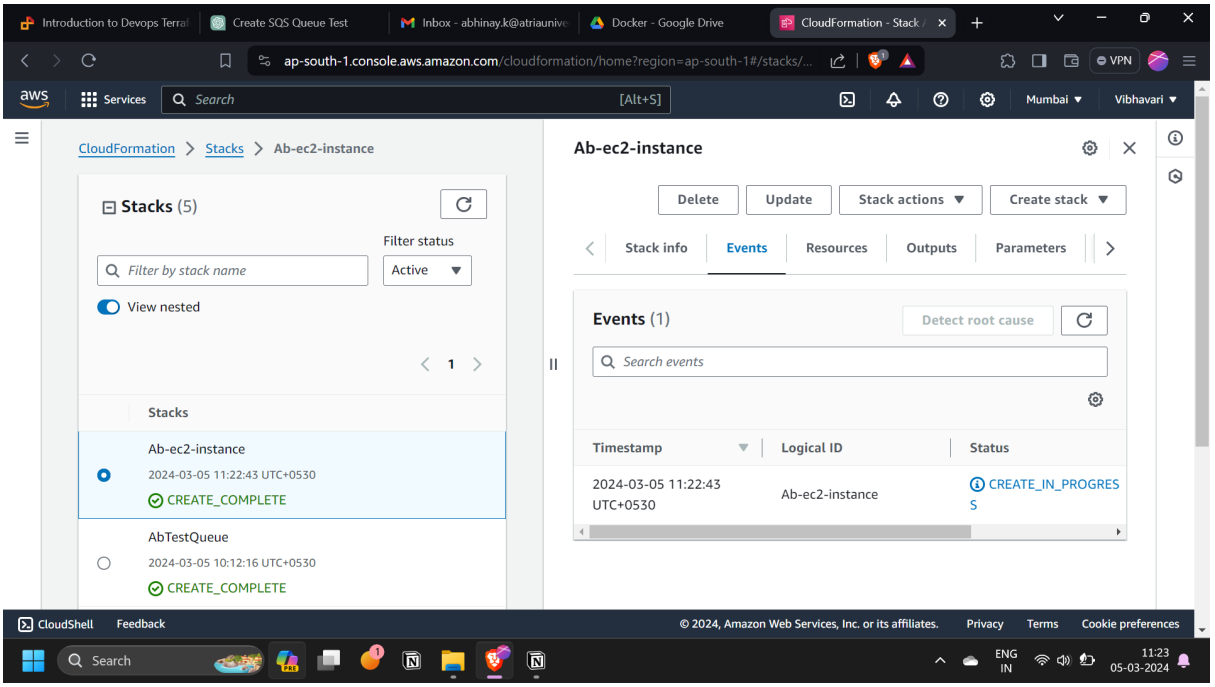
Step 5: Reviewing and Creating the Stack:
Review the stack configuration and click on "Create stack" to initiate the provisioning process.



Step 6: Monitoring Stack Creation:

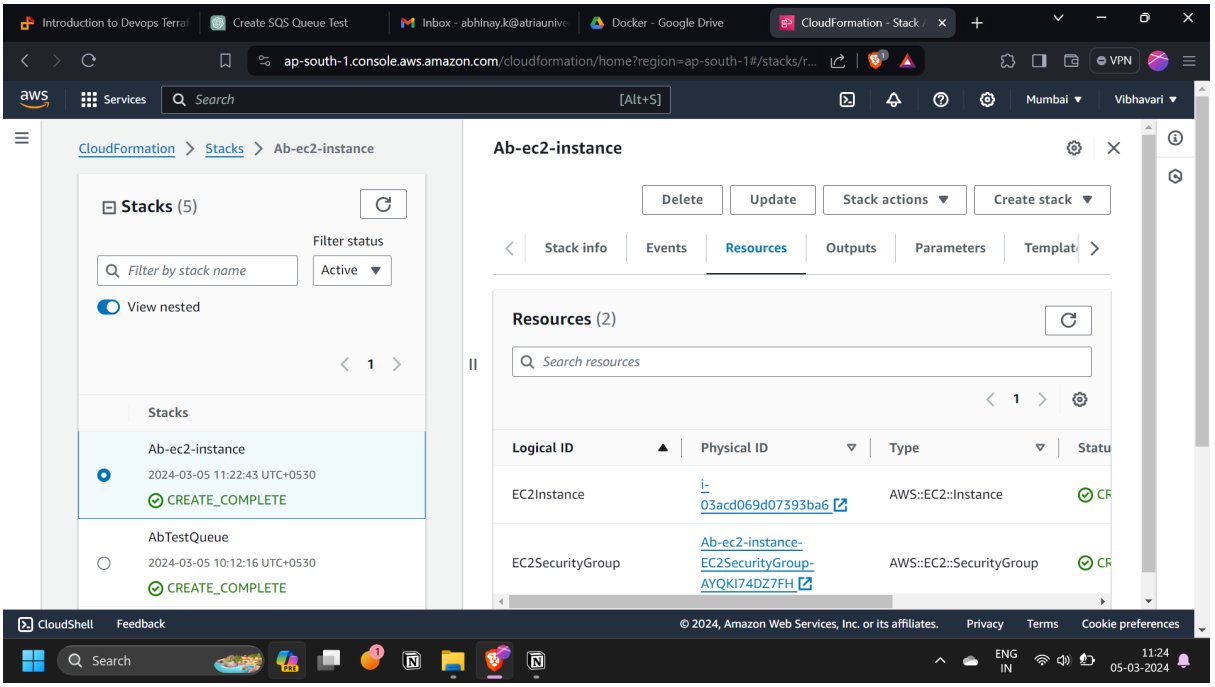
Navigate to the "Stacks" section within CloudFormation and monitor the progress of stack creation. Wait until the status changes to "CREATE_COMPLETE".

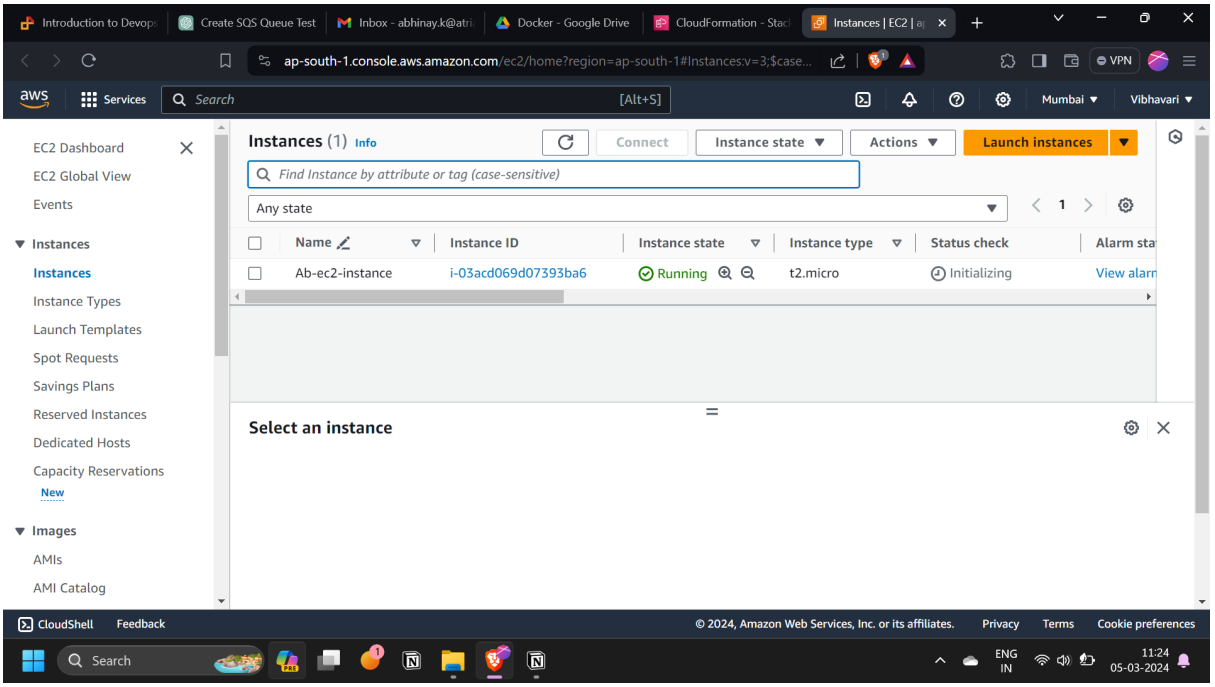




Step 7: Accessing EC2 Instance:

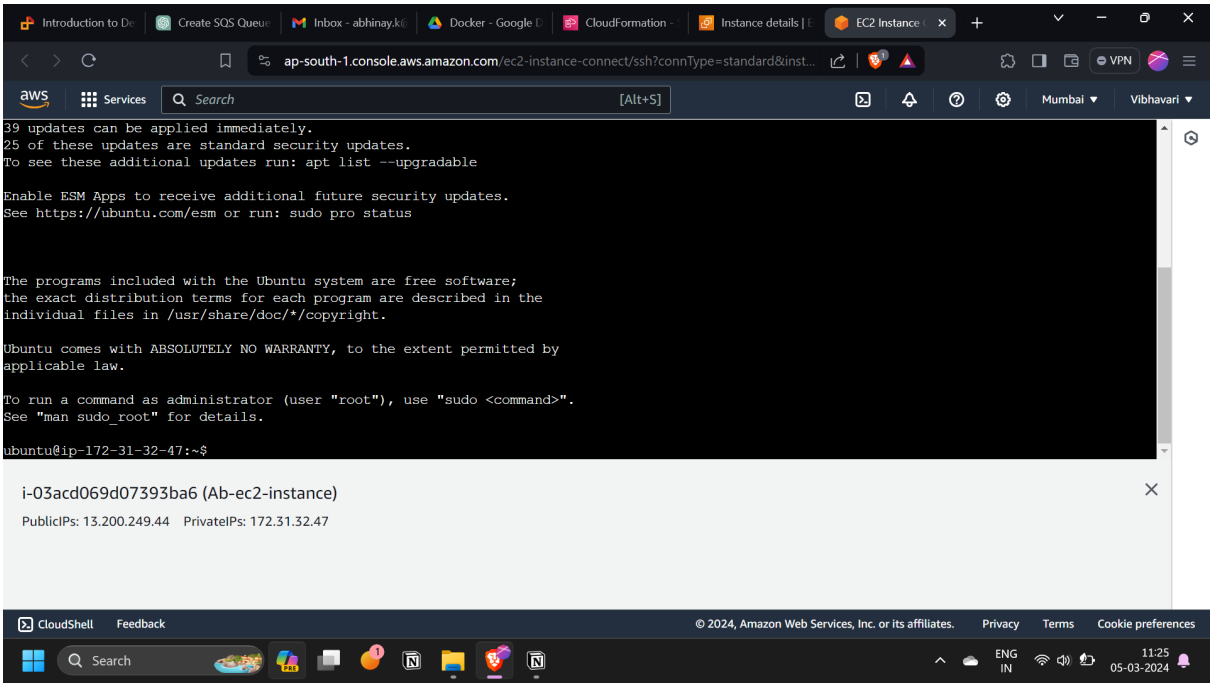
Once the stack creation is complete, navigate to the EC2 service from the AWS Management Console. Locate the newly created EC2 instance.

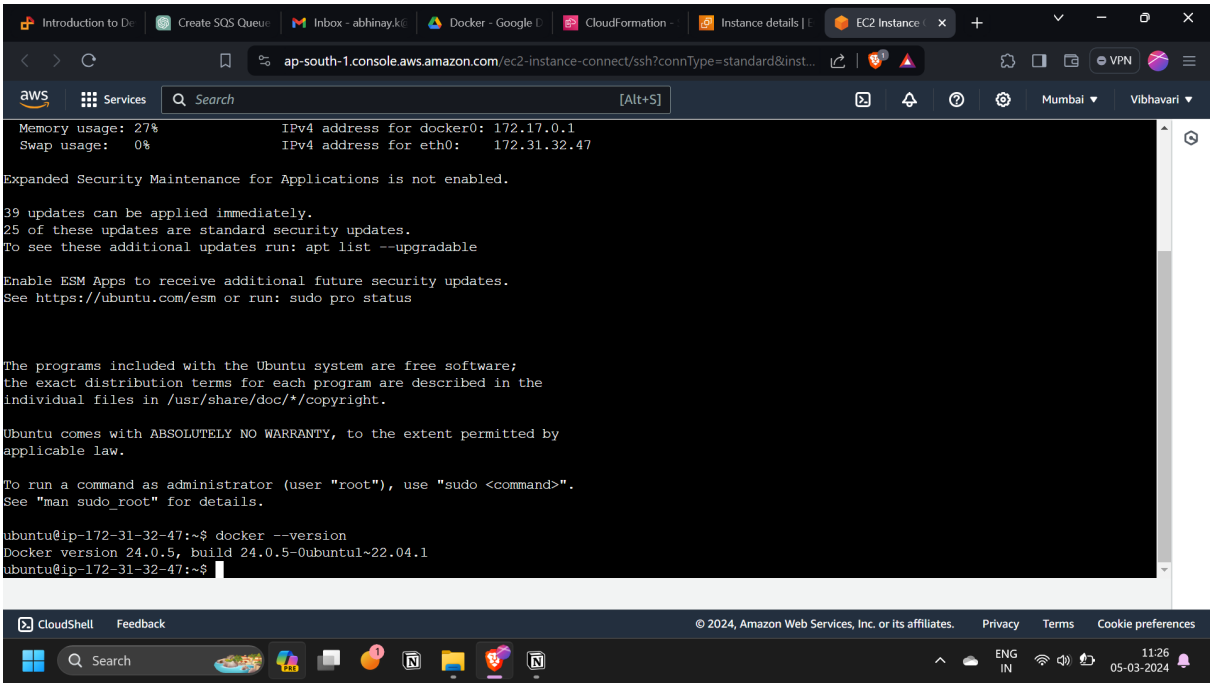




Step 8: Verifying Docker Installation:

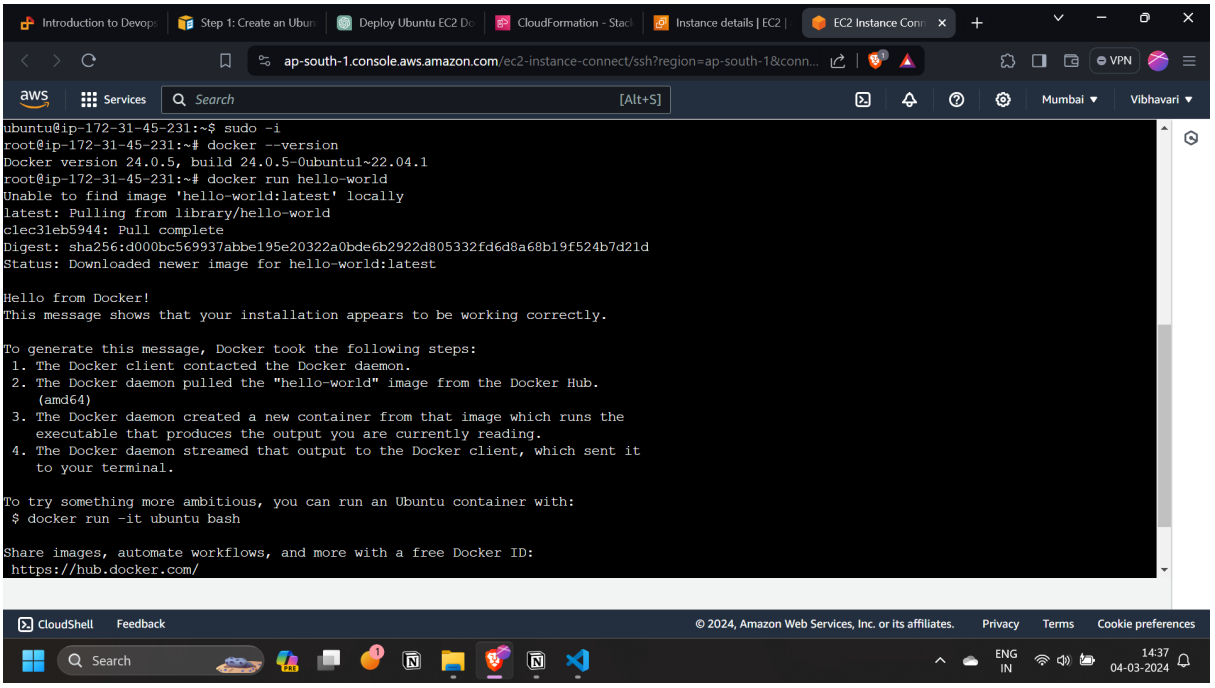
Connect to the EC2 instance using SSH and verify that Docker is installed properly by running the command `docker --version`.





Step 9: Testing Docker:

Run a simple Docker command, such as `docker run hello-world`, to ensure that Docker is functioning correctly on the EC2 instance.



Conclusion

The Ubuntu EC2 instance with Docker installed has been successfully provisioned using AWS CloudFormation, providing a platform for further development and deployment of TELEMAX's architecture.

Task 2 : Create SQS Queue

Create a SQS Queue and Test with Dummy Data: Manually

1. **Create a SQS Queue:**
    - Go to the AWS Management Console and navigate to the Amazon SQS service.
    - Click on "Create Queue".
    - Enter a name for your queue and configure any additional settings as needed (such as message retention period, visibility timeout, etc.).
    - Click "Create Queue" to create the SQS queue.
2. **Test with Dummy Data:**
    - After creating the SQS queue, navigate to the "Send and receive messages" section.
    - Click on "Send a message".
    - Enter some dummy data (e.g., a JSON message) and click "Send message".
    - Verify that the message is successfully sent to the queue.

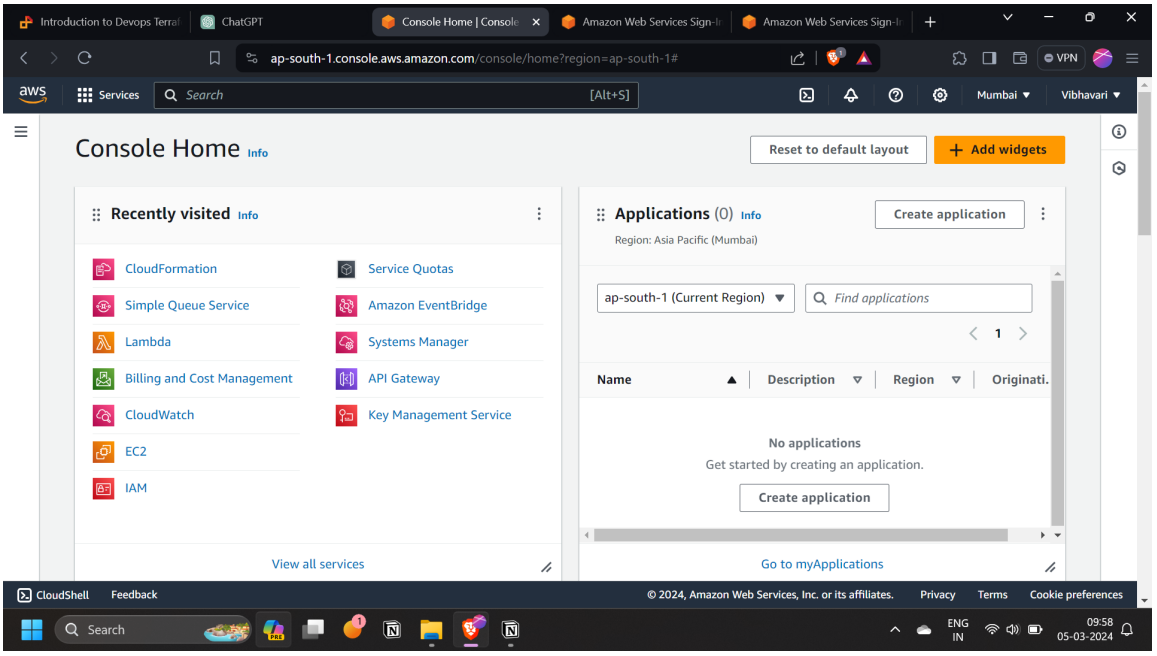
Create a SQS Queue and Test with Dummy Data with CloudFormation Template

Create a CloudFormation Template:

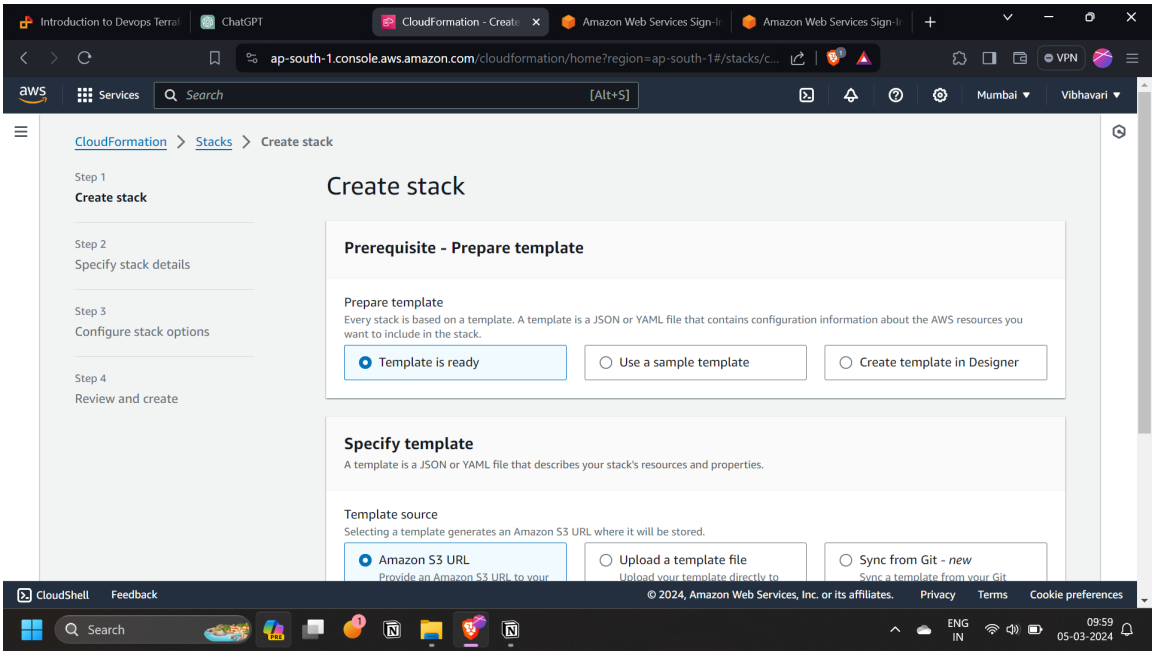
Template code

```
AWSTemplateFormatVersion: '2010-09-09'
Resources:
  MyQueue:
    Type: "AWS::SQS::Queue"
    Properties:
      QueueName: "AbTestQueue"

Outputs:
  QueueURL:
    Description: "URL of the SQS queue"
    Value: !GetAtt AbTestQueue.QueueUrl
```

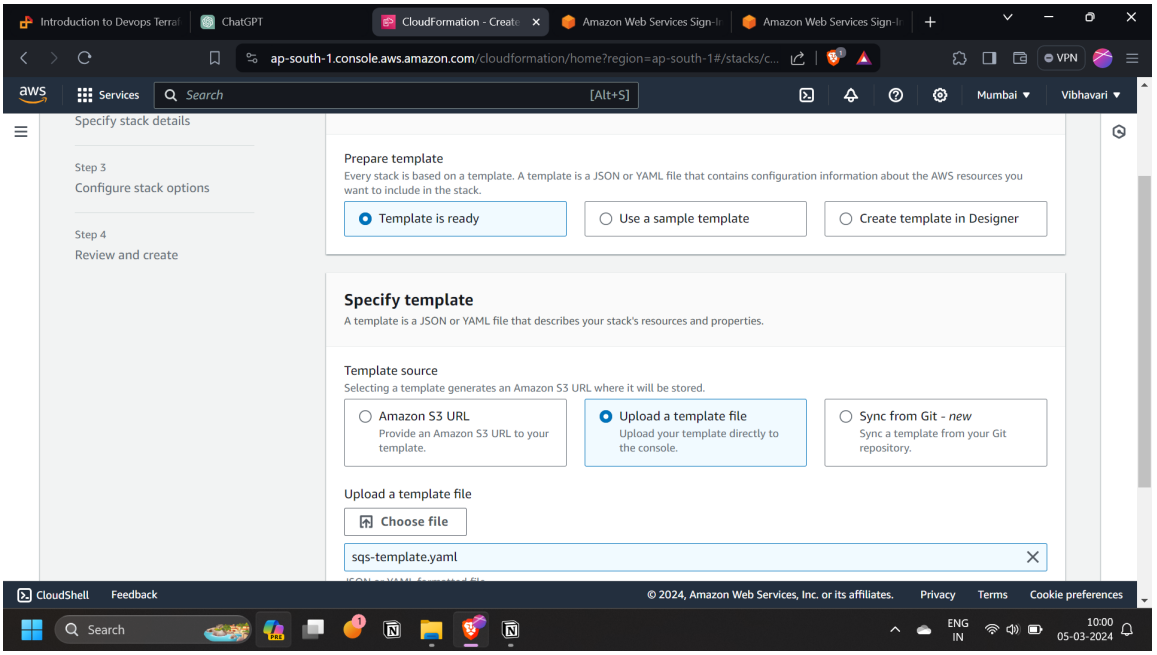
- Log in to the AWS Management Console and navigate to the CloudFormation service.



- Click on "Create Stack" to create a new stack.

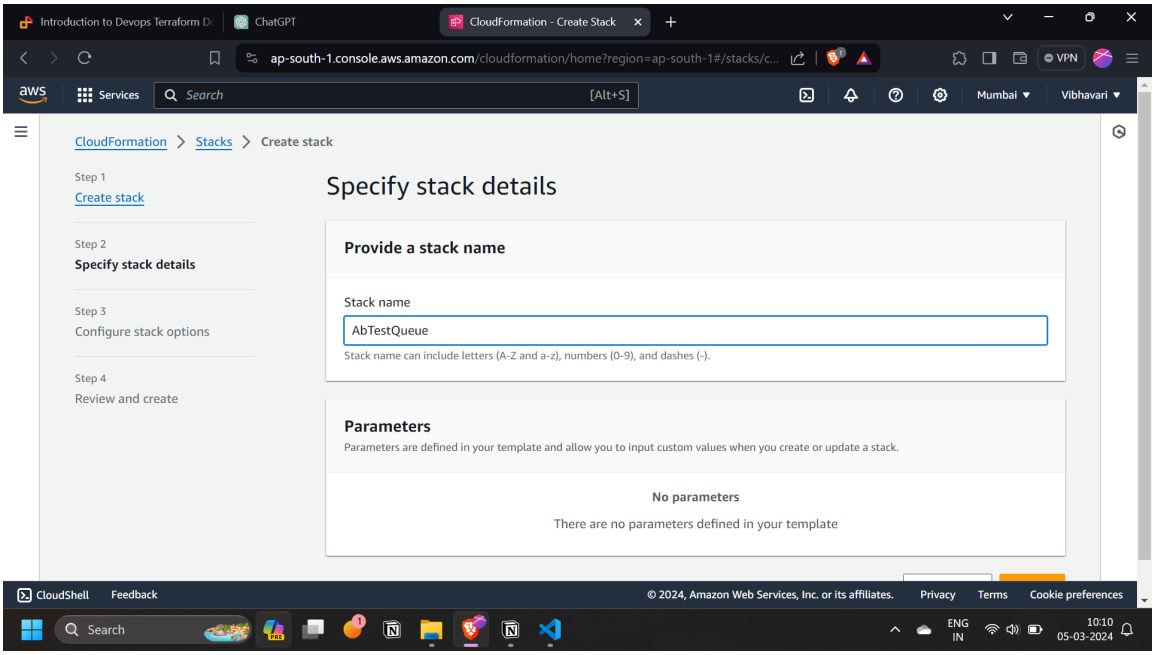


- Select "Template is ready" and "Upload a template file".

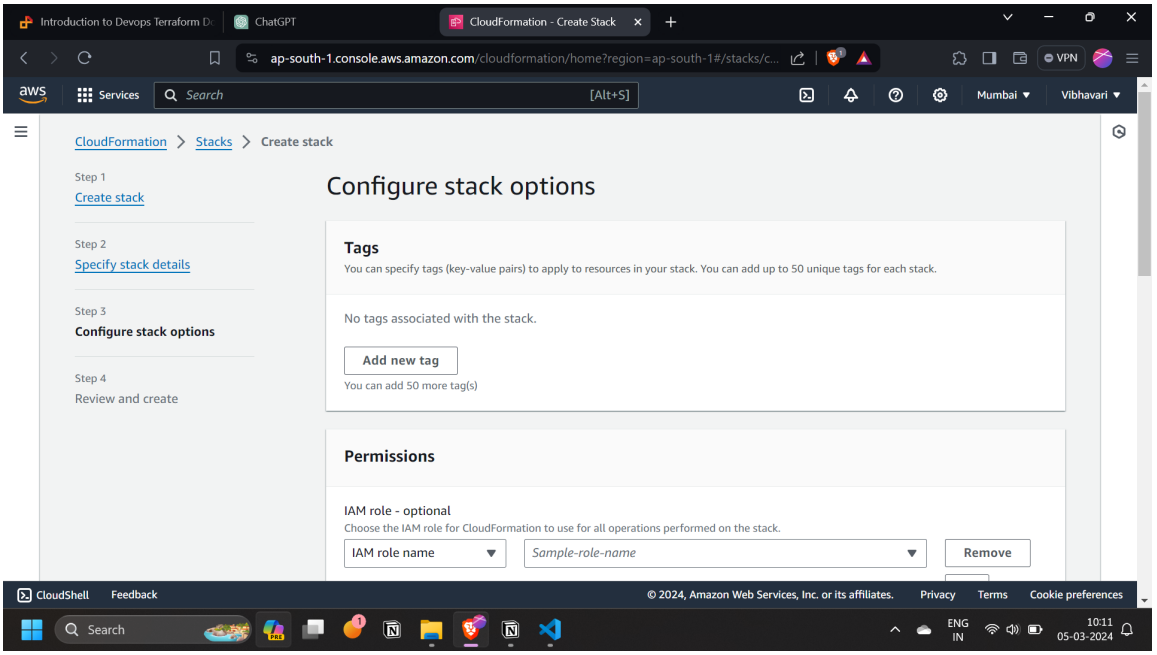


Deploy the CloudFormation Stack:

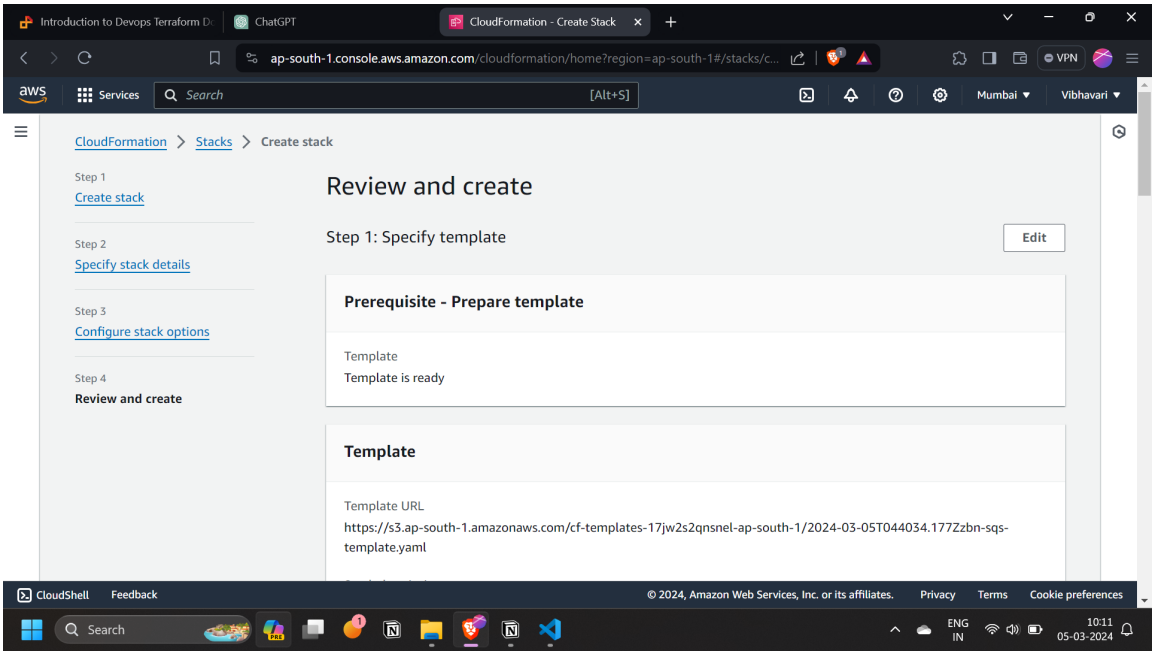
- Click "Next" to proceed to the stack creation wizard.



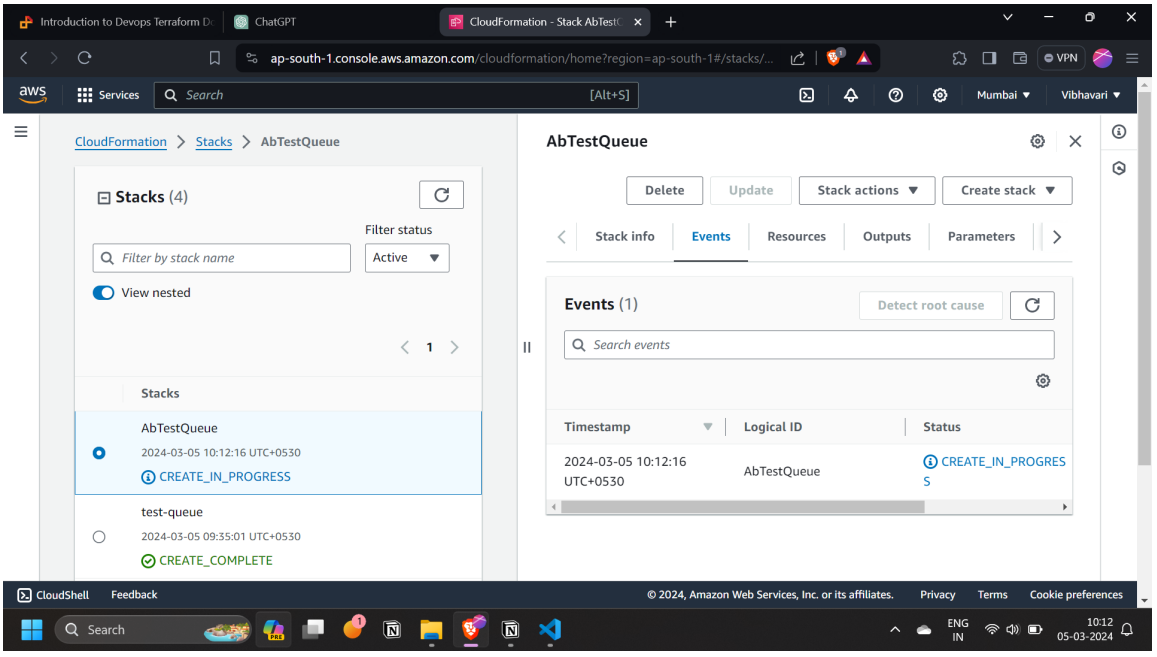
- Enter a stack name and any other parameters as needed.



- Click "Next" to proceed to the review page.

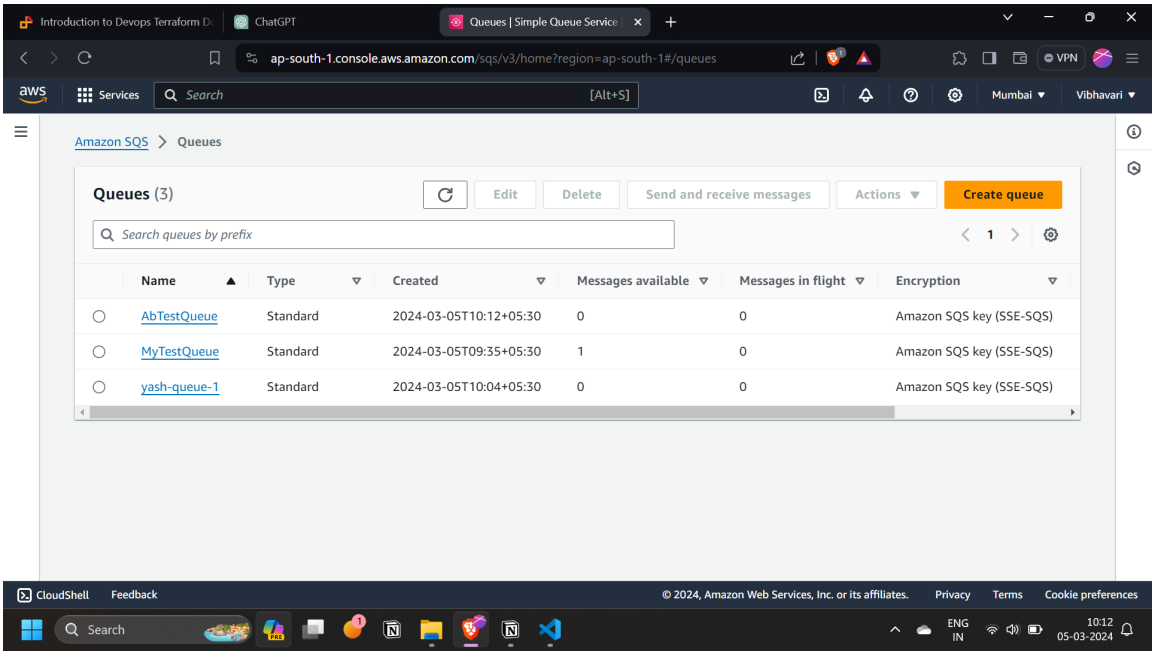


- Review the stack details and click "Create stack" to deploy the CloudFormation stack.

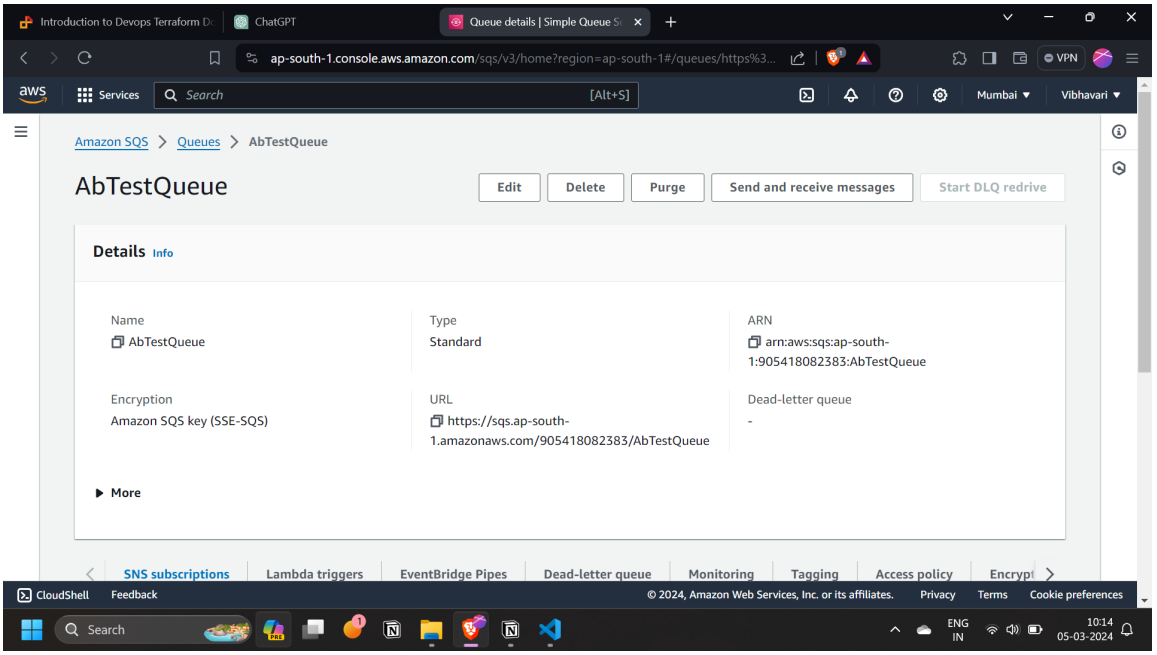


Test with Dummy Data:

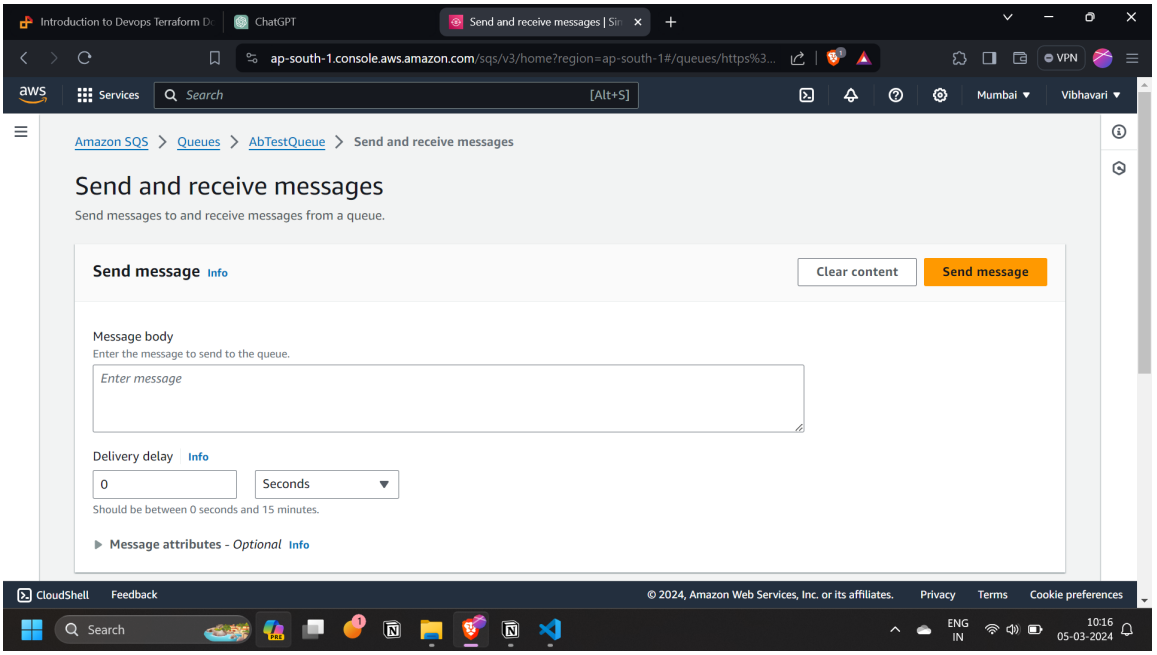
- After the stack creation is complete, navigate to the SQS service in the AWS Management Console.



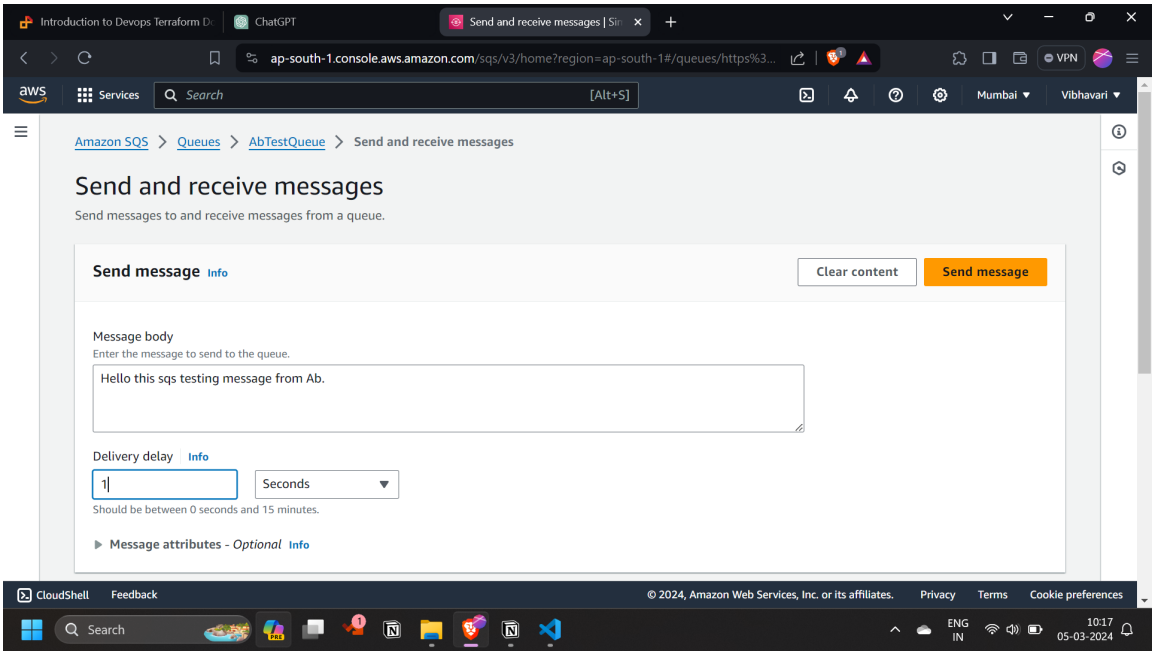
- Find the SQS queue created by the CloudFormation stack



- Click on "Send a message".

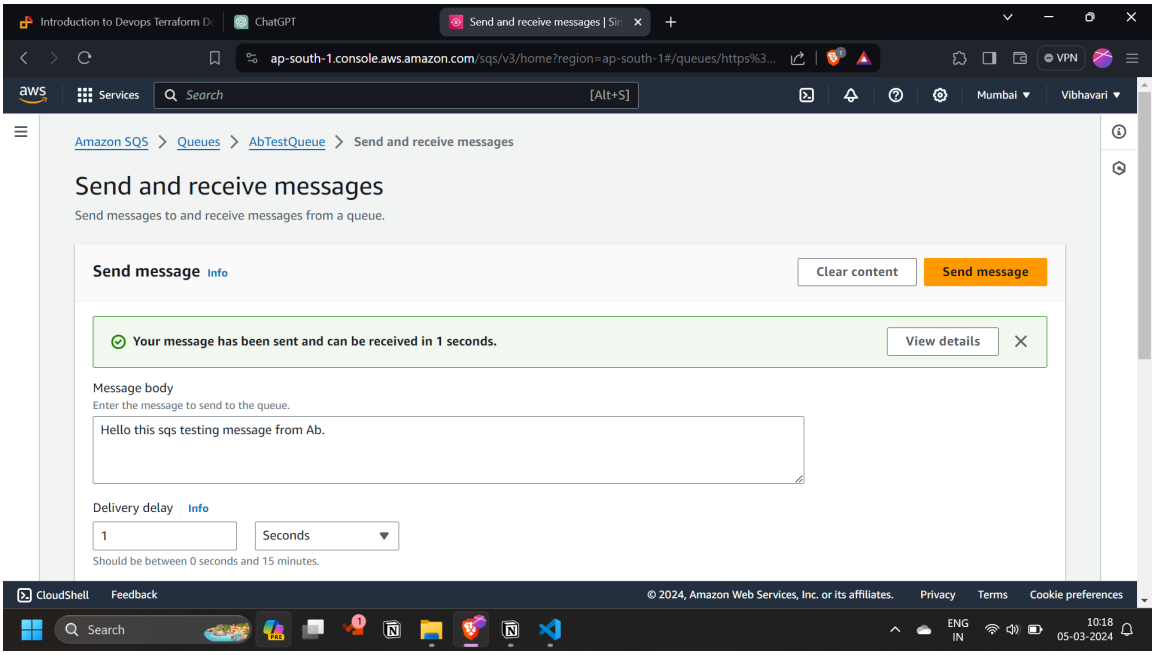


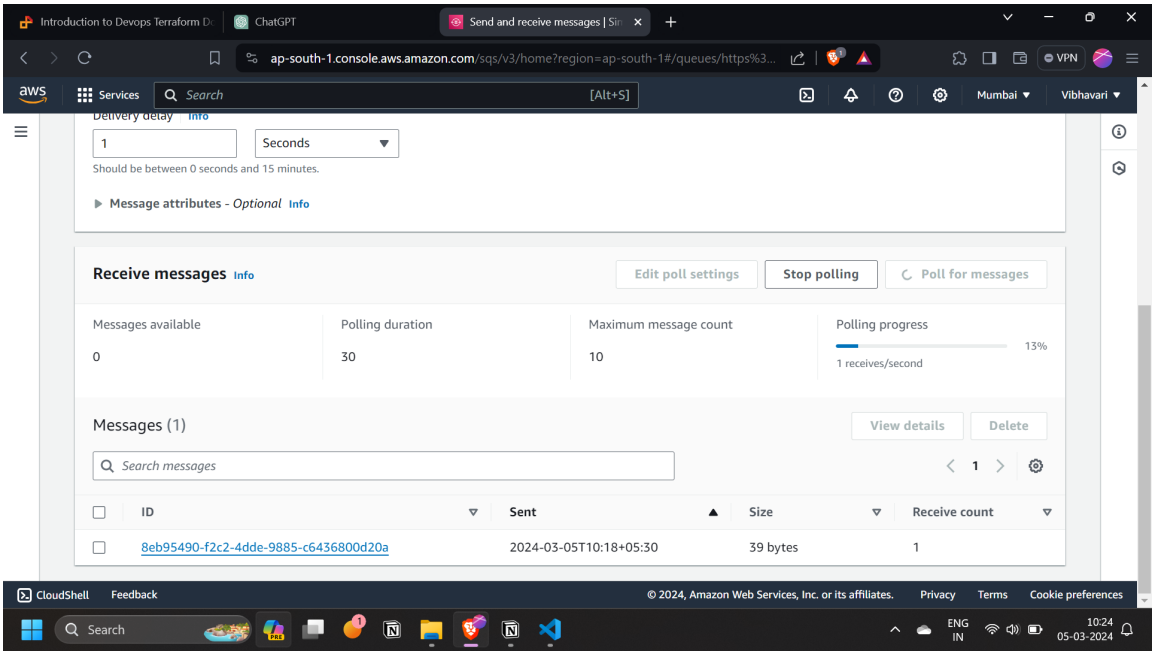
- Enter some dummy data and click "Send message".

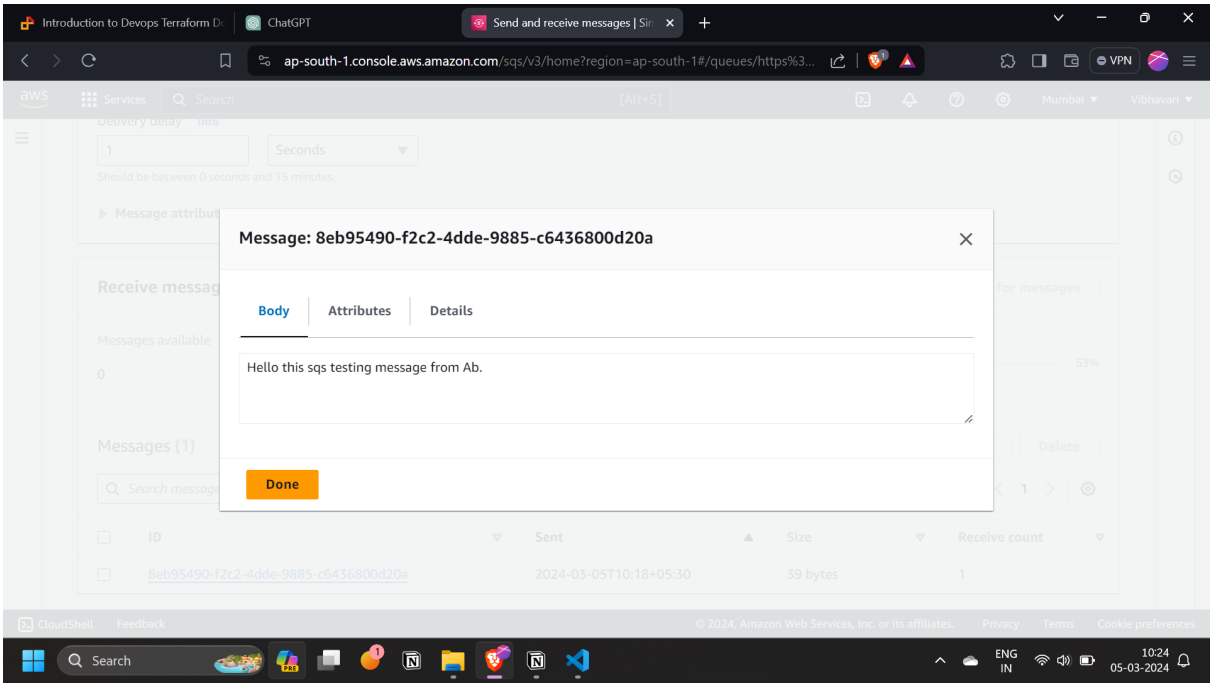


Verify message

- Verify that the message is successfully sent to the queue.


Introduction to Devops Terraform D
ChatGPT
Send and receive messages | Sin
ap-south-1.console.aws.amazon.com/sqs/v3/home?region=ap-south-1#/queues/https%3...
VPN
aws
Services
Search
[Alt+S]
Mumbai
Vibhavari
Amazon SQS > Queues > AbTestQueue > Send and receive messages
Send and receive messages
Send messages to and receive messages from a queue.
Send message Info
Clear content
Send message
Your message has been sent and can be received in 1 seconds.
View details
Message body
Enter the message to send to the queue.
Hello this sqs testing message from Ab.
Delivery delay Info
1
Seconds
Should be between 0 seconds and 15 minutes.
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Search
ENG
IN
10:18
05-03-2024


Introduction to Devops Terraform D
ChatGPT
Send and receive messages | Sin
ap-south-1.console.aws.amazon.com/sqs/v3/home?region=ap-south-1#/queues/https%3...
VPN
aws
Services
Search
[Alt+S]
Mumbai
Vibhavari
Delivery delay Info
1
Seconds
Should be between 0 seconds and 15 minutes.
Message attributes - Optional Info
Receive messages Info
Edit poll settings
Stop polling
Poll for messages
Messages available
0
Polling duration
30
Maximum message count
10
Polling progress
13%
1 receives/second
Messages (1)
View details
Delete
Search messages
1
ID
Sent
Size
Receive count
8eb95490-f2c2-4dde-9885-c6436800d20a
2024-03-05T10:18+05:30
39 bytes
1
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences
Search
ENG
IN
10:24
05-03-2024



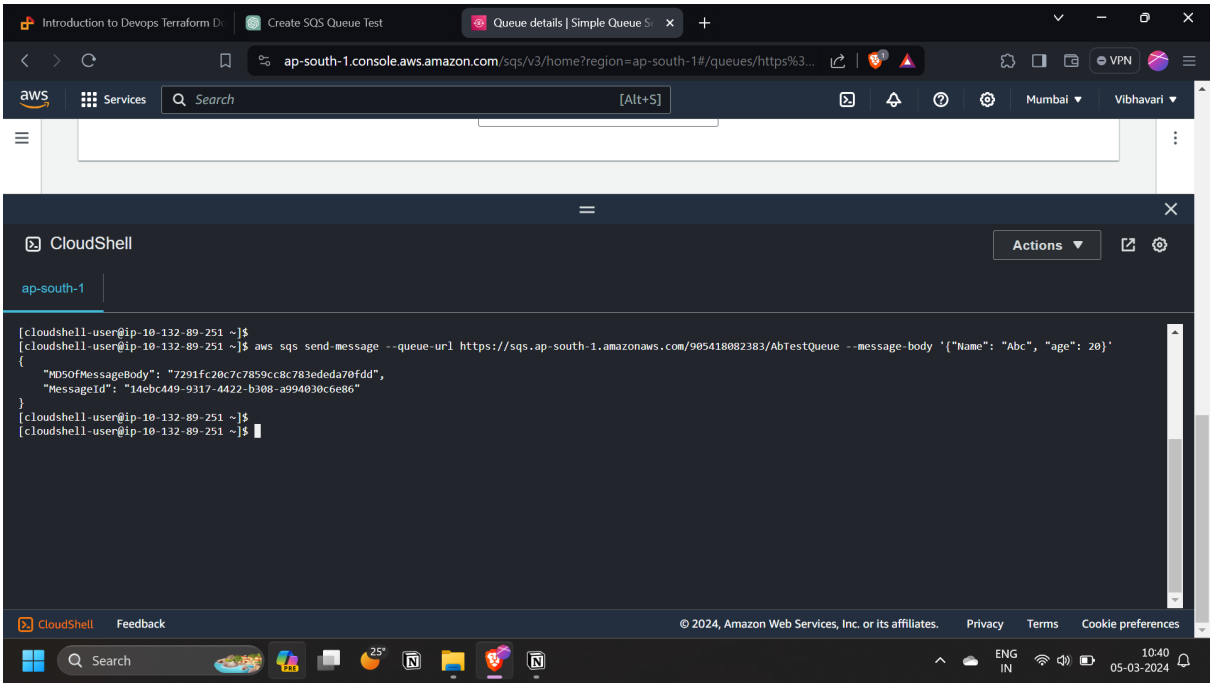
Testing Dummy Data using AWS CLI

Sending dummy data to the queue using the AWS CLI:

Command

```
aws sqs send-message --queue-url <https://sqs.ap-south-1.amazonaws.com/905418082383/AbTestQueue> --message-body '{"Name": "Abc", "age": 20}'
```

Screenshot

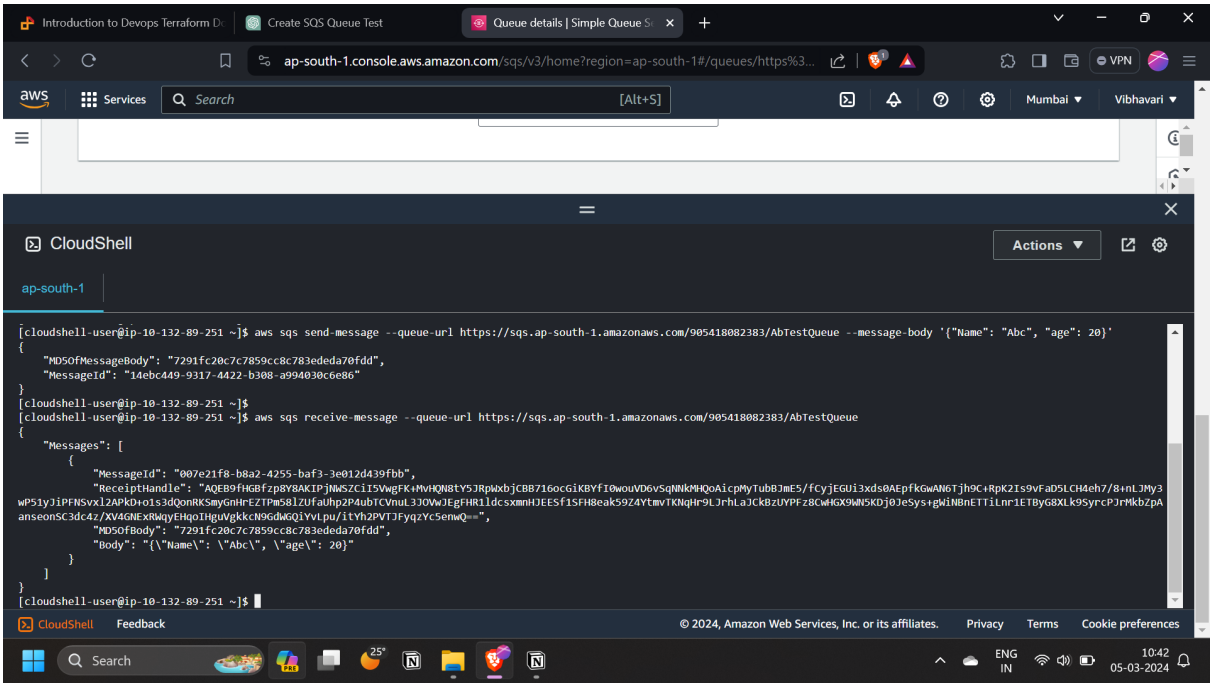


View the message in AWS CLI:

Command:

```
aws sqs receive-message --queue-url <https://sqs.ap-south-1.amazonaws.com/905418082383/AbTestQueue>
```

Screenshot



Task 3: Install Docker on the EC2 Instance

1. **Connect to the EC2 Instance**:
   - Log in to the AWS Management Console.
   - Navigate to the EC2 service.
   - Select the EC2 instance that you previously created.
   - Click on "Connect" to view connection instructions.
   - Follow the instructions to connect to the instance using SSH.
2. **Update the Package Repository**:
   - Once connected to the EC2 instance, update the package repository to ensure you install the latest version of Docker.

```
sudo apt update
```

3. **Install Docker**:
   - Install Docker CE (Community Edition) using the package manager.

```
sudo apt install docker.io
```

4. **Start and Enable Docker Service**:
   - Start the Docker service and enable it to start on boot.

```
sudo systemctl start docker
sudo systemctl enable docker
```

5. **Verify Docker Installation**:
   - Check the Docker version to ensure that it's installed correctly.

```
docker --version
```

6. **Test Docker**:
   - Run a simple Docker command to verify that Docker is installed and working properly.

```
sudo docker run hello-world
```

This command should download and run the "hello-world" Docker image. If everything is set up correctly, you should see a message confirming that Docker is working.

Conclusion

By completing these steps, you have successfully installed Docker on the EC2 instance. This will allow you to create and manage Docker containers on your instance, which can be useful for various development and deployment tasks.

Task 4 Create IAM Role with Limited Access

Create IAM Role with Limited Access (ECR Permissions should be allowed)

Description: This subtask involves creating an IAM role that grants limited permissions, specifically allowing access to Amazon Elastic Container Registry (ECR) resources.

Here's a CloudFormation template to create an IAM role with limited access, specifically granting permissions for Amazon ECR (Elastic Container Registry):

```
AWSTemplateFormatVersion: '2010-09-09'
Resources:
  ECRRole:
```

```
    Type: AWS::IAM::Role
    Properties:
      RoleName: ECRRole
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: ec2.amazonaws.com
            Action: sts:AssumeRole
      Policies:
        - PolicyName: ECRPolicy
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: Allow
                Action:
                  - ecr:GetAuthorizationToken
                  - ecr:BatchCheckLayerAvailability
                  - ecr:GetDownloadUrlForLayer
                  - ecr:GetRepositoryPolicy
                  - ecr:DescribeRepositories
                  - ecr:ListImages
                  - ecr:DescribeImages
                  - ecr:BatchGetImage
                Resource: '*'
```

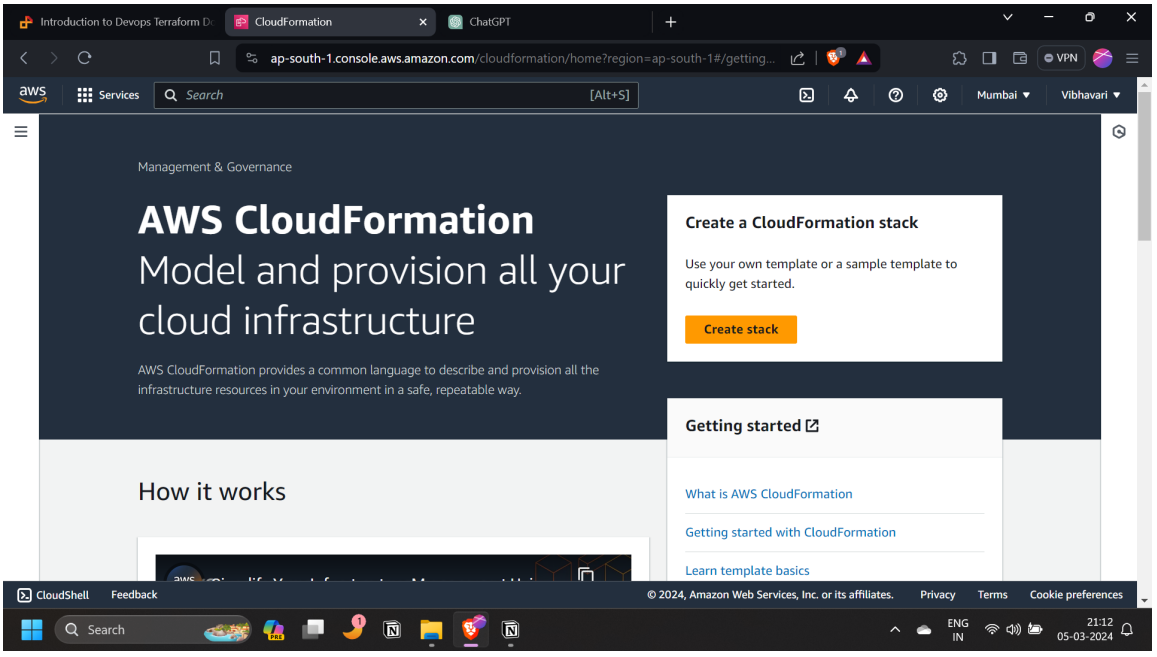
Explanation:

- This CloudFormation template defines an IAM role named `ECRRole`.
- The `AssumeRolePolicyDocument` specifies that only EC2 instances (`ec2.amazonaws.com`) are allowed to assume this role.
- The `ECRPolicy` grants permissions for various actions within Amazon ECR, such as `GetAuthorizationToken`, `BatchCheckLayerAvailability`, etc.
- The `Resource: '*'` allows access to all resources within ECR. You can further restrict this to specific repositories if needed.

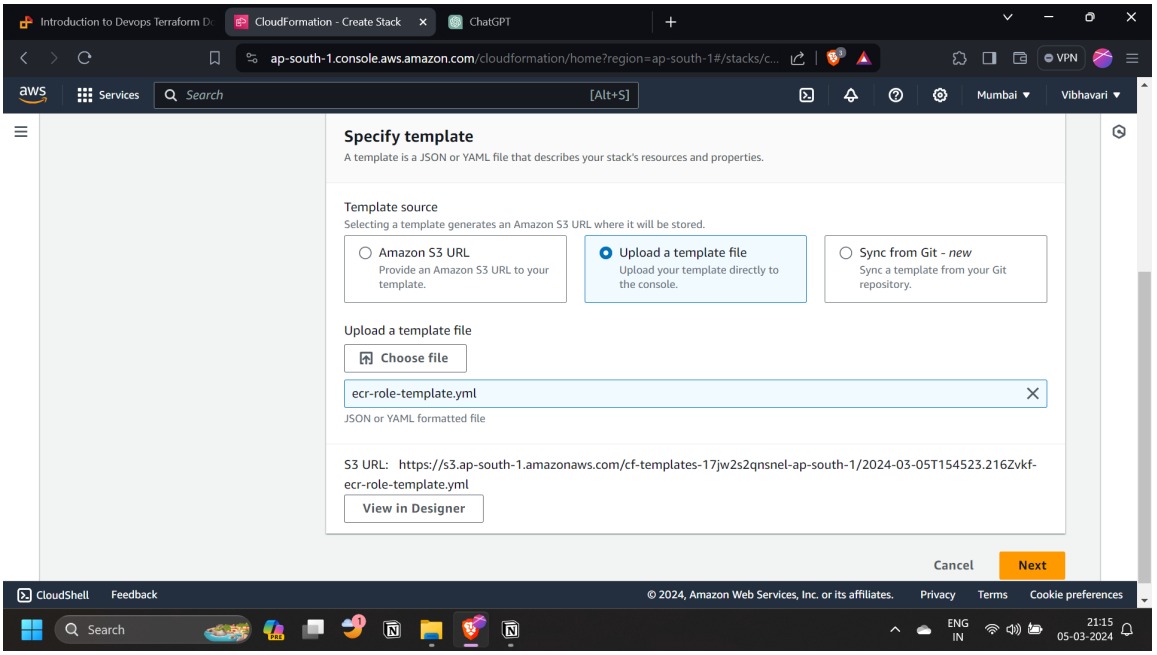
We will deploy this template using the AWS Management Console, AWS CLI, or any other AWS SDK that supports CloudFormation. Make sure to replace the `RoleName` with a suitable name for your use case.

Implement the IAM role with limited access for Amazon ECR using the provided CloudFormation:

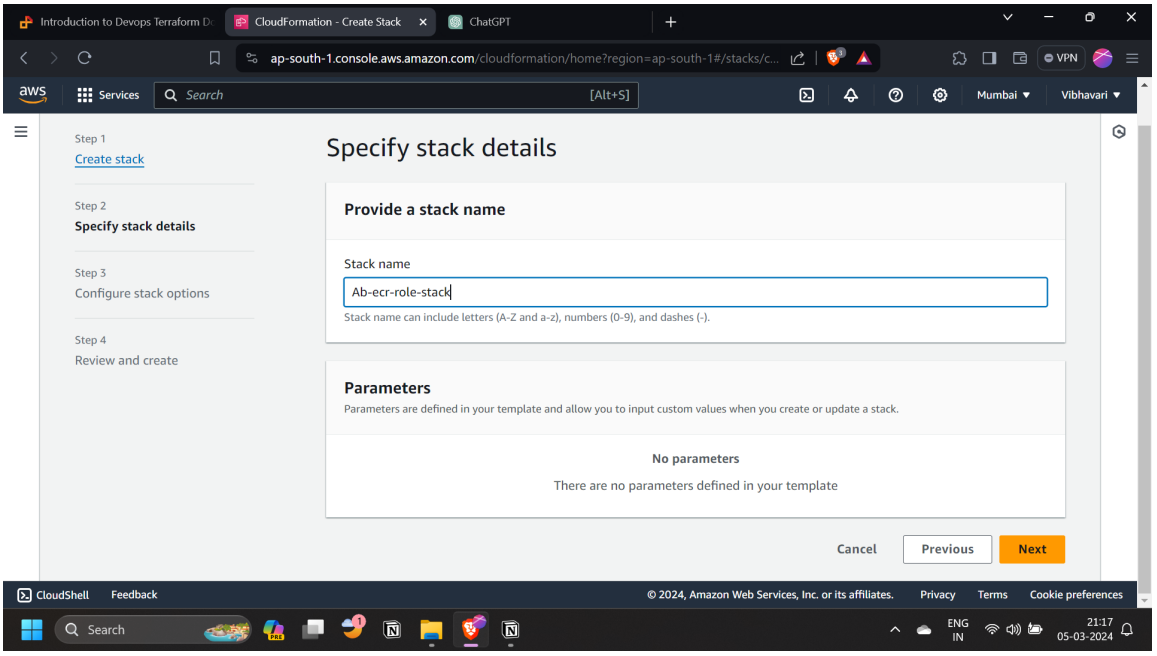
1. **Save the CloudFormation Template**:
   Save the CloudFormation template provided in the previous response to a file with a `.yaml` or `.json` extension. You can name it something like `ecr-role-template.yaml`.
2. **Navigate to CloudFormation**:
   Go to the AWS CloudFormation service by either searching for it in the AWS services search bar or navigating through the services dropdown.



3. **Select Template**:
   Choose "Template is ready" and then select "Upload a template file". Click on "Choose file" and select the CloudFormation template file (`ecr-role-template.yaml`) you saved earlier.

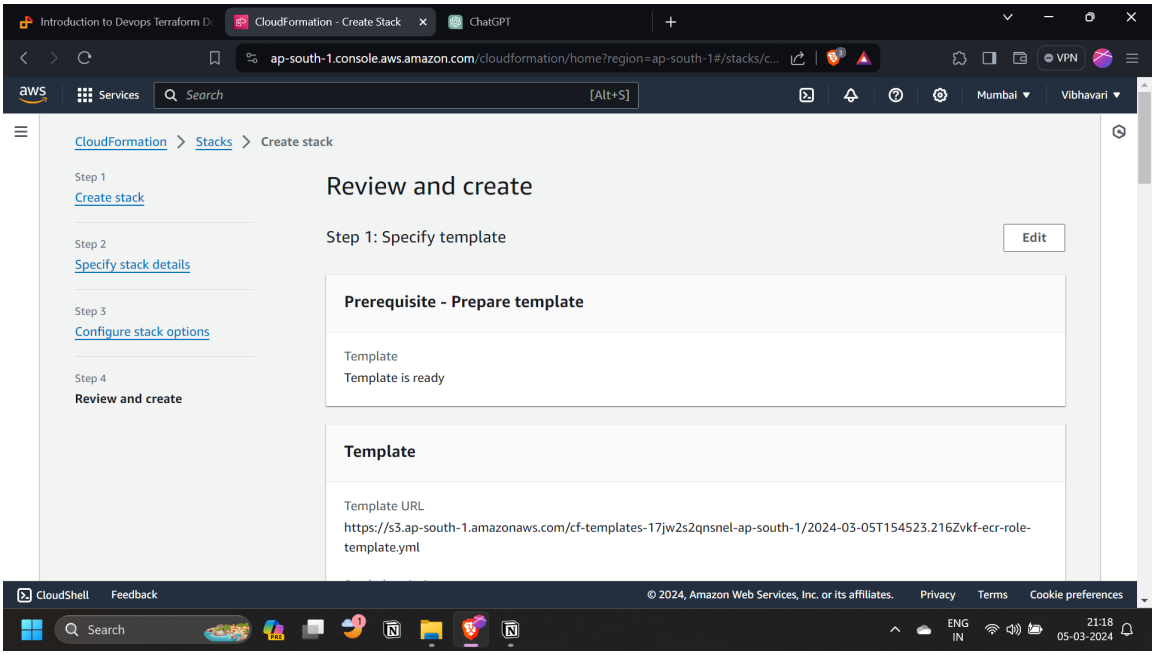


4. **Specify Stack Details**:
   Provide a stack name, for example, `ECRRoleStack`.

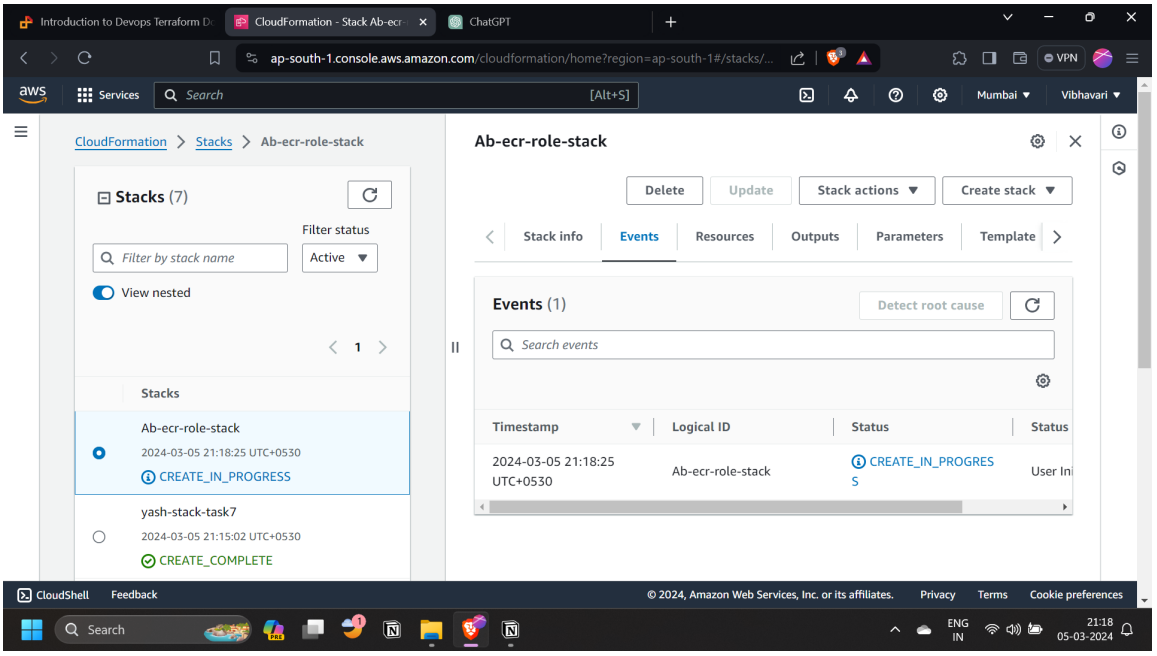


5. **Configure Stack Options** (Optional):
   You can specify additional configurations such as tags, permissions, etc., if needed. Otherwise, you can leave these as default.

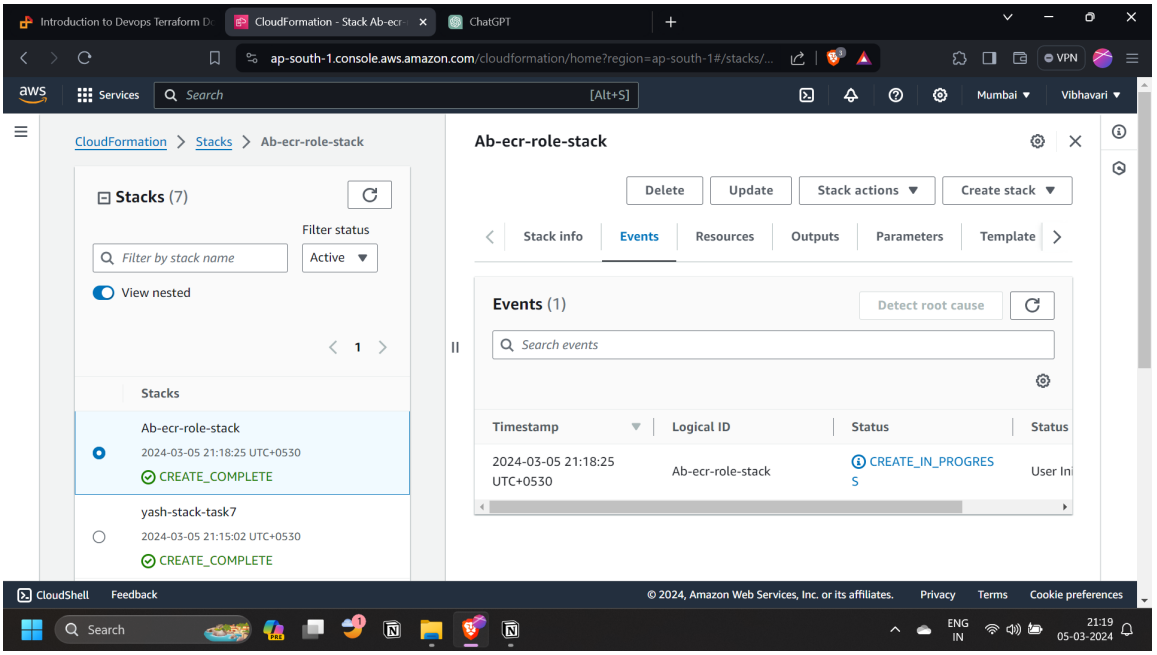
6. **Review and Create**:
   Review the stack details and click on "Create stack" to initiate the creation process.



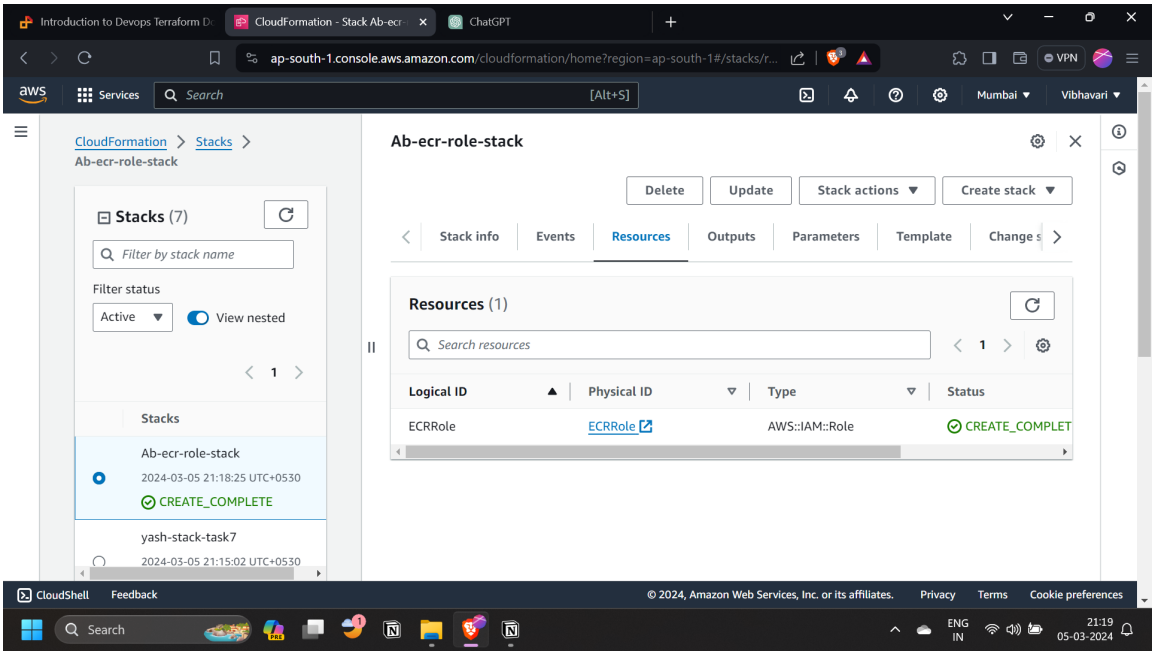
7. **Wait for Stack Creation**:
   Wait for the CloudFormation stack creation process to complete. This usually takes a few minutes.

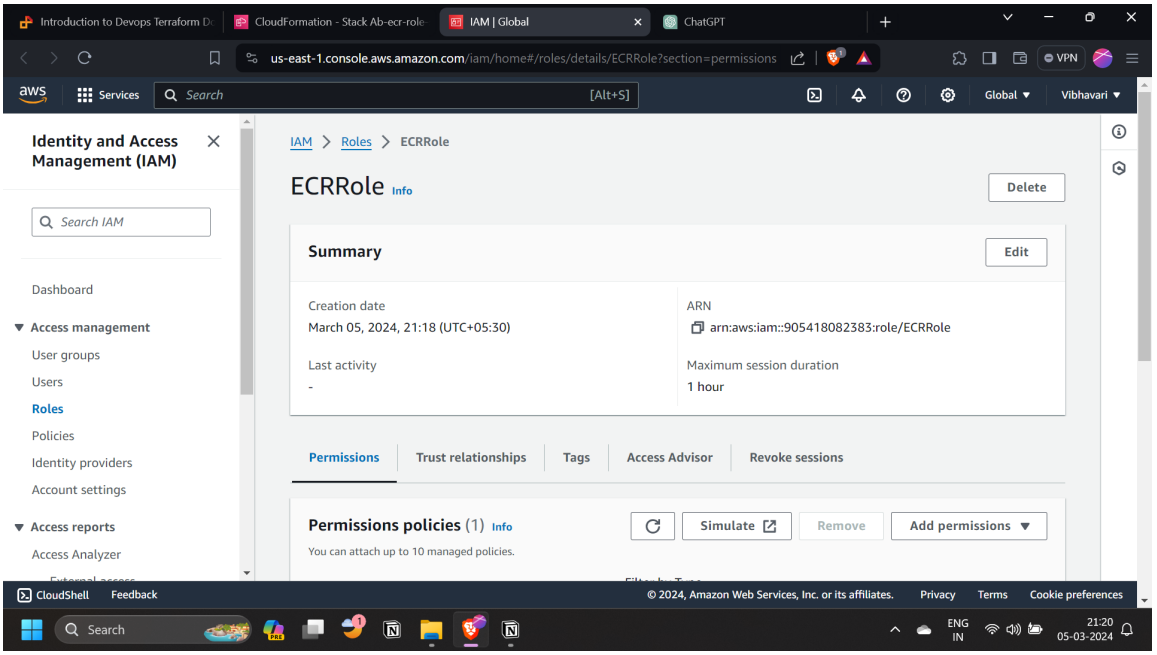


8. **Verify Stack Creation**:
   Once the stack creation is complete, navigate to the "Stacks" section in the CloudFormation console. Verify that the stack status is "CREATE_COMPLETE".



9. **Access the IAM Role**:
   You can now access the IAM role named `ECRRole` that you've created. This role grants limited access to Amazon ECR resources as defined in the CloudFormation template.





Conclusion:

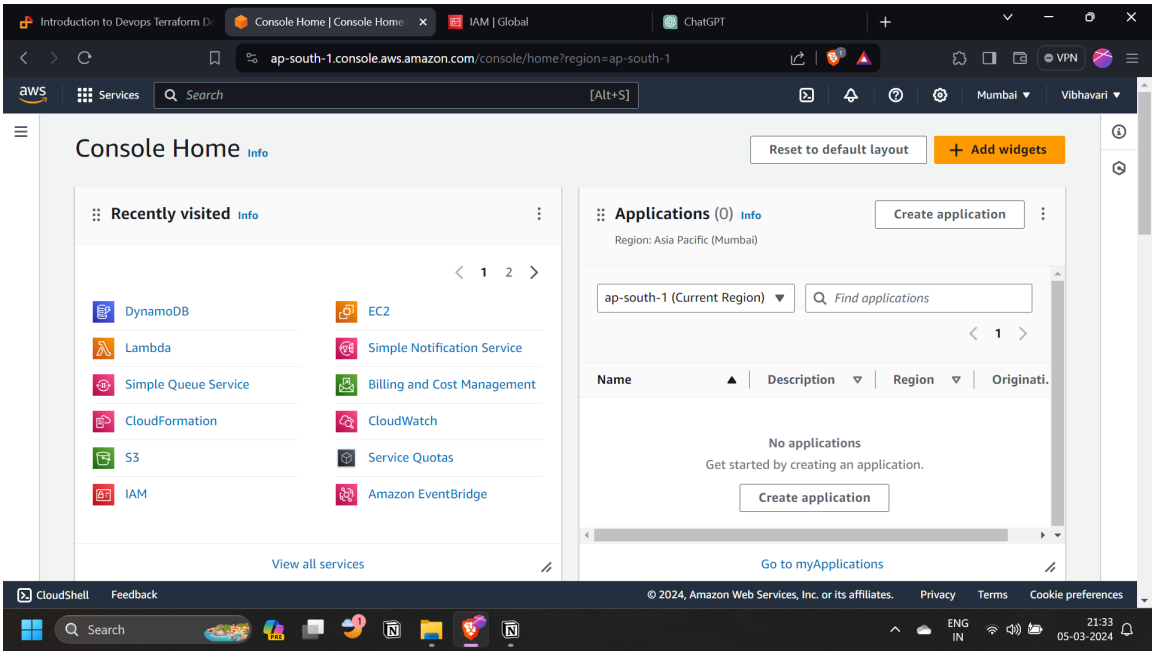
Implemented an IAM role with limited access for Amazon ECR using CloudFormation. This role, named ECRRole, grants specific permissions to interact with Amazon ECR resources, ensuring security and compliance in your AWS environment. We can now leverage this role for various use cases such as managing container images, deploying applications, or integrating with other AWS services, all while adhering to a least privilege access model.

Task 5: Attach IAM role to EC2 instance

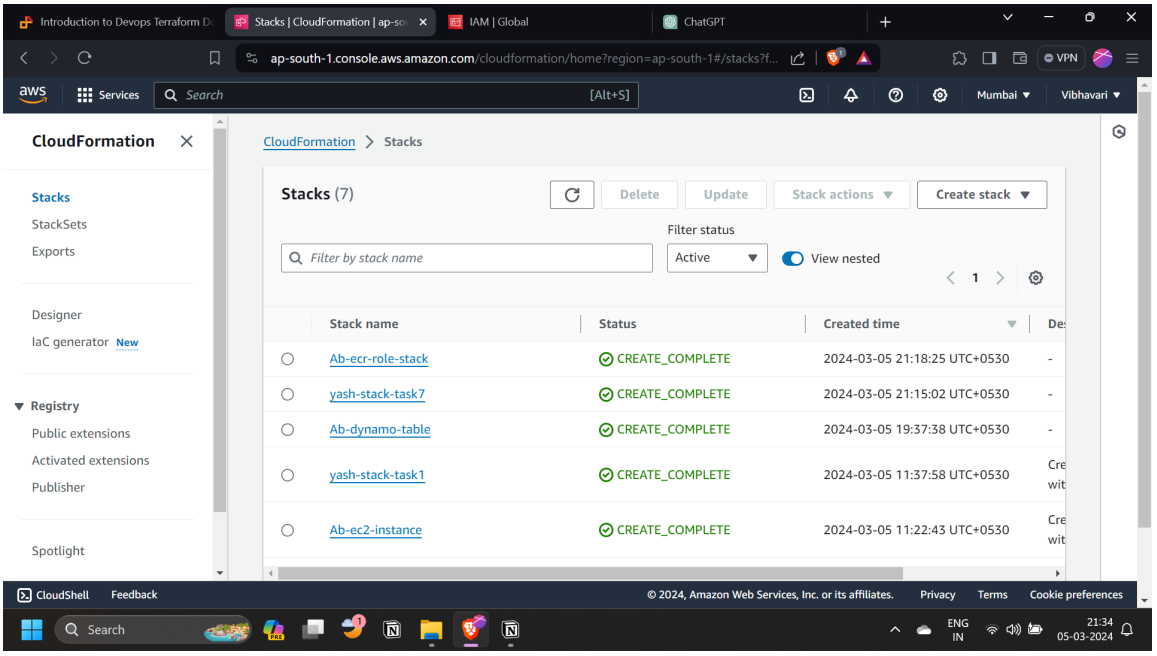
Attach IAM role to above created instance in Task 1.

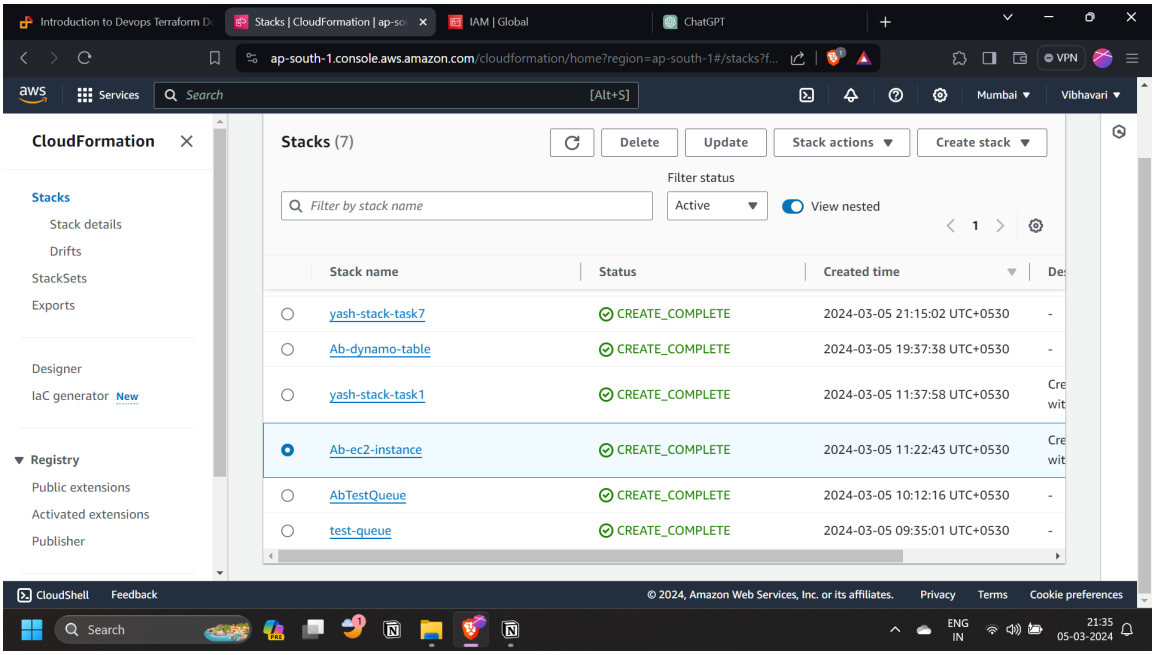
Implementation of attaching an IAM role to an existing EC2 instance using the AWS Management Console:

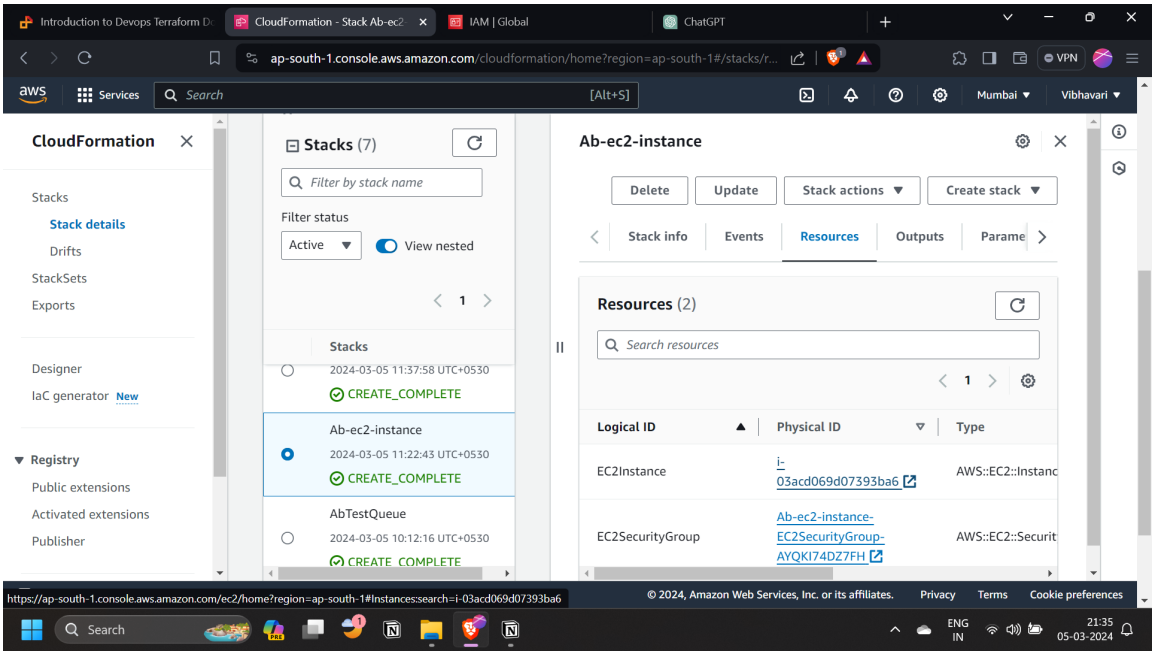
1. **Sign in to the AWS Management Console**: Go to the AWS Management Console and sign in using your credentials.



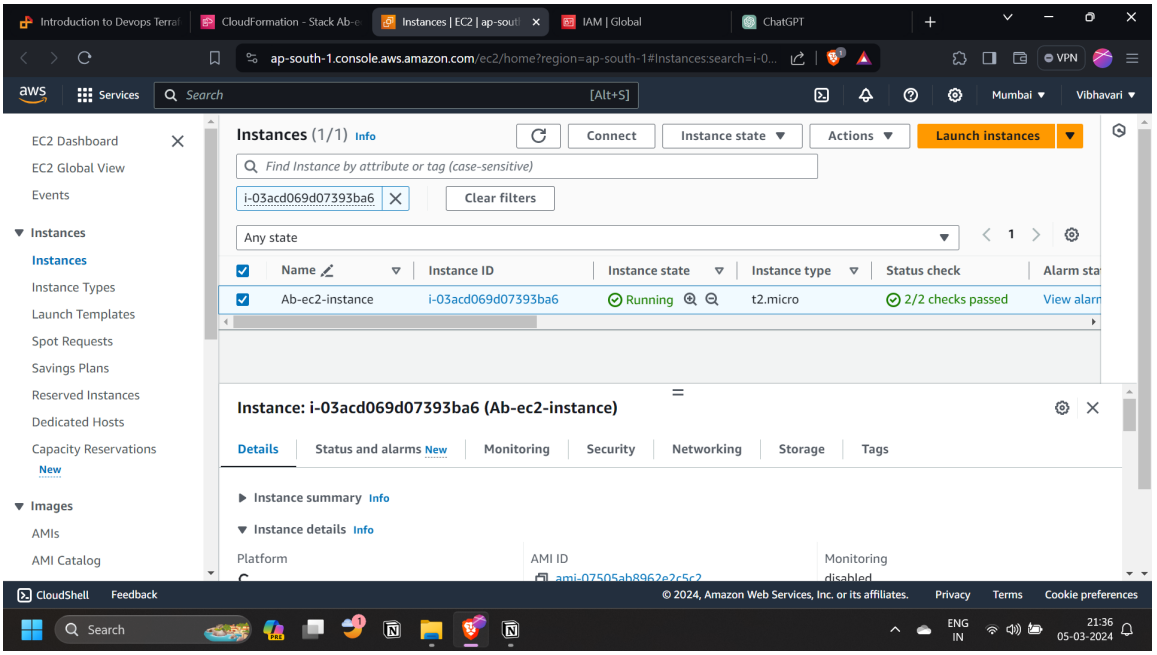
2. **Navigate to the EC2 Dashboard**: Once logged in, navigate to the EC2 dashboard by searching for "EC2" in the AWS services search bar and clicking on it.

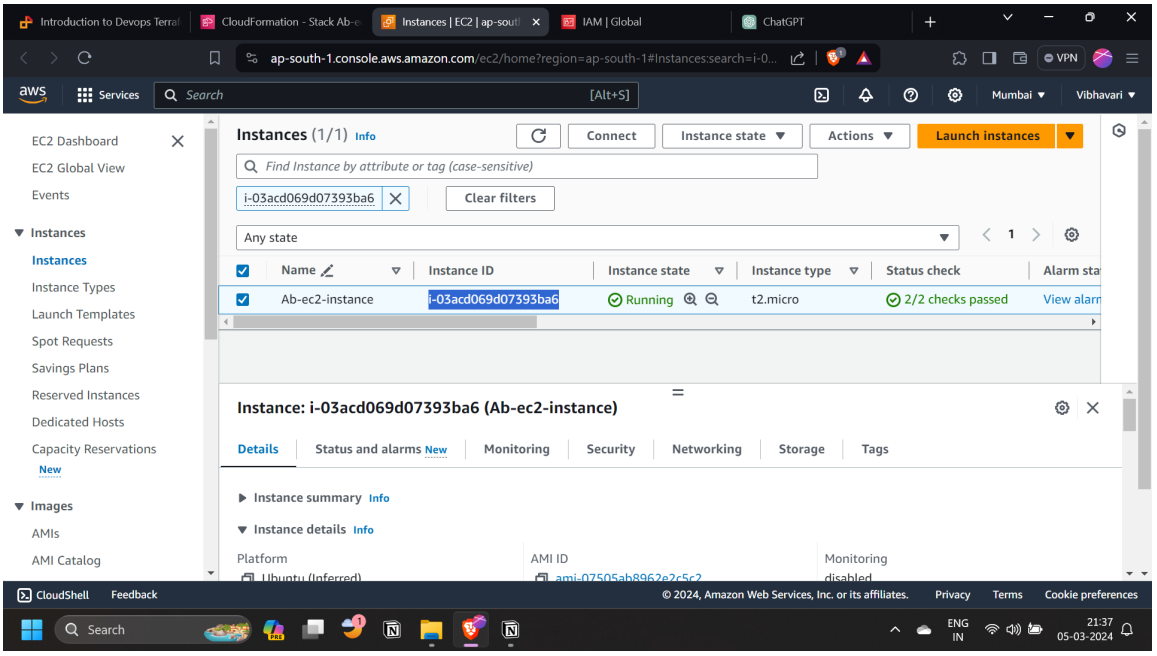




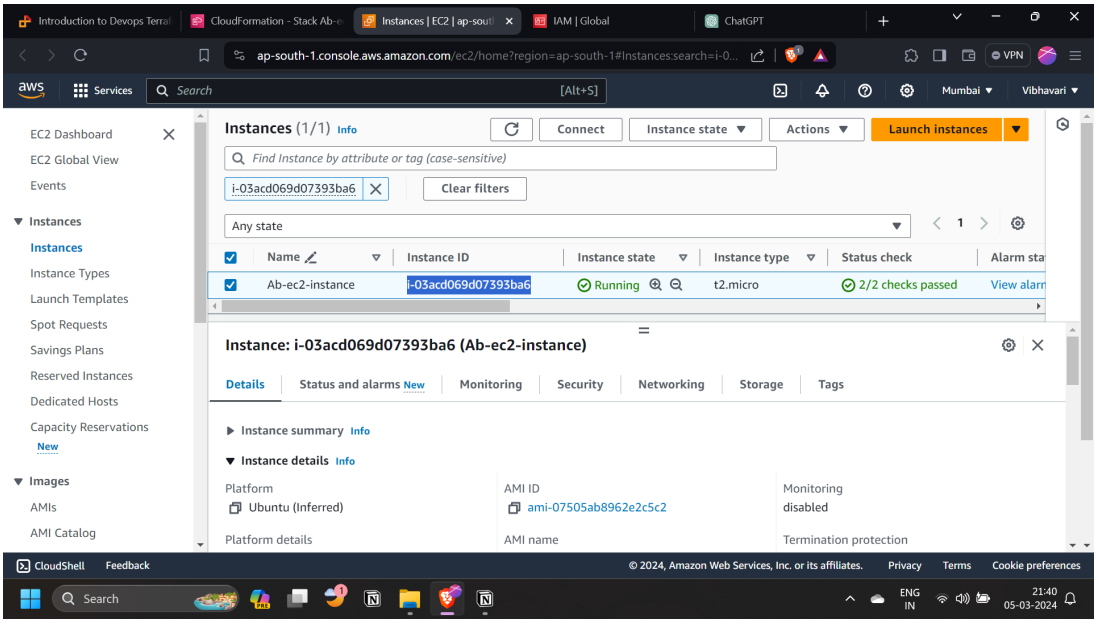


3. **Select the EC2 Instance**: From the EC2 dashboard, locate and select the Ubuntu EC2 instance that you created in Task 1. Click on the instance ID to select it.

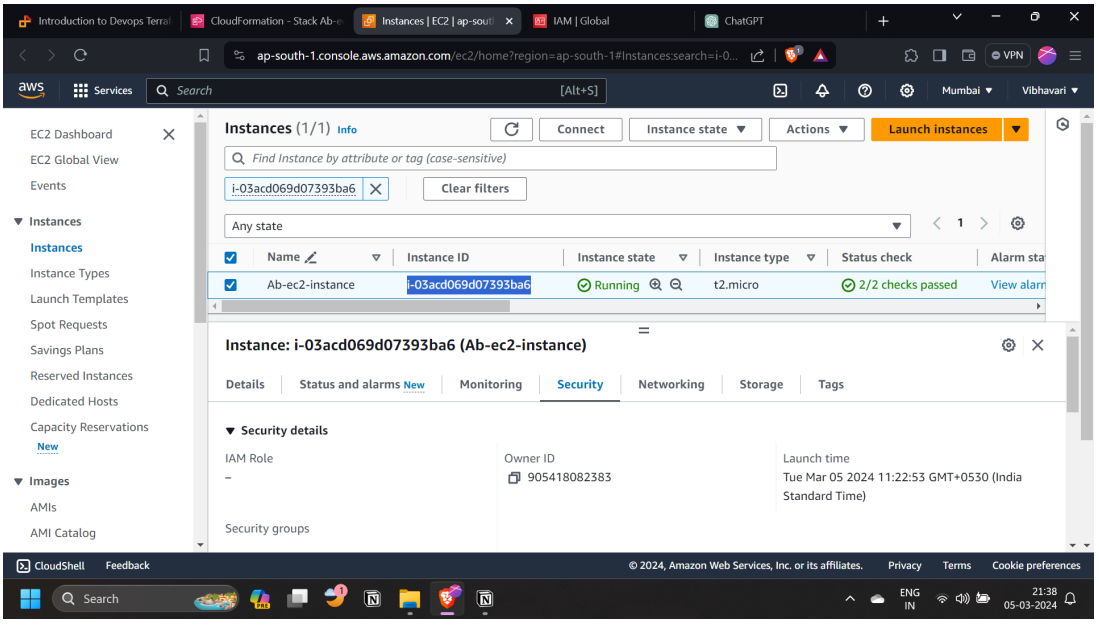




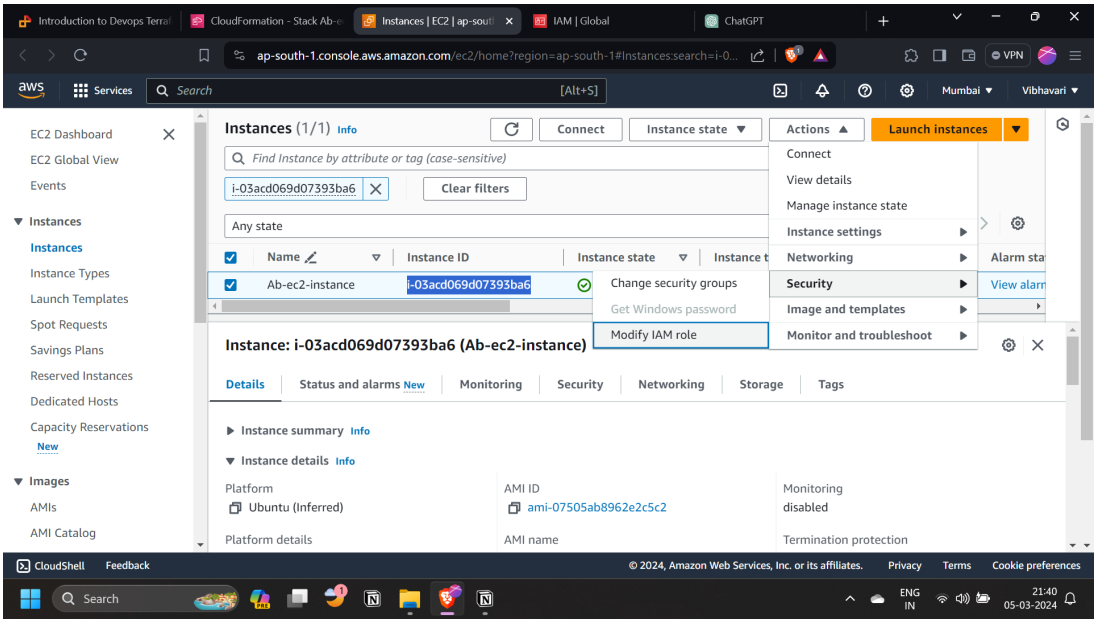
4. **Attach IAM Role**:
    - With the EC2 instance selected, scroll down to the "Description" tab at the bottom of the page.



- In the "Details" section, find the "IAM role" field, which initially may show "None".



- Click on the "Actions" dropdown button and select "Security" > "Modify IAM role".



- In the "Modify IAM role" dialog, you'll see a dropdown list of available IAM roles. Select the IAM role named `ECRRole` that you created in Task 4.
- Click on the "Apply" button to attach the IAM role to the EC2 instance.

5. **Confirmation**:
   - After attaching the IAM role, you'll receive a confirmation message indicating that the IAM role has been successfully attached to the EC2 instance.
6. **Verify Attachment**:
   - To verify that the IAM role has been attached, you can select the EC2 instance again and check the "IAM role" field in the "Description" tab. It should now display the name of the attached IAM role (`ECRRole`).

That's it! You have successfully attached the IAM role named `ECRRole` to the Ubuntu EC2 instance created earlier. Now the instance has the necessary permissions to interact with Amazon ECR, as defined by the IAM role.

Task 6: Create DynamoDB Table

Creating DynamoDB table with AWS Management Console:

Using AWS Management Console:

- Go to the AWS Management Console and navigate to the DynamoDB service.
- Click on "Create table".
- Enter the table name (e.g., YourTableName).
- Under "Primary key", enter "ProductID" as the partition key and select "String" as its type.
- Add additional attributes by clicking on "Add sort key" or "Add index" depending on your requirements.
- Click on "Create".

Using AWS CloudFormation:

Below is a CloudFormation template to create a DynamoDB table:

```
Resources:
  ProductTable:
    Type: 'AWS::DynamoDB::Table'
    Properties:
      TableName: Product
      AttributeDefinitions:
        - AttributeName: productId
          AttributeType: S
      KeySchema:
        - AttributeName: productId
          KeyType: HASH
      ProvisionedThroughput:
        ReadCapacityUnits: 5
        WriteCapacityUnits: 5
```

To deploy this template:

- Save the above YAML content in a file, say `dynamodb-table.yaml`.
- Go to the AWS CloudFormation service in the AWS Management Console.
- Click on "Create stack" and upload the YAML file.

- Follow the prompts to create the stack, providing necessary inputs like stack name.
- Once the stack creation is complete, your DynamoDB table will be created according to the specified configuration.

Implementation of Creating Dynamodb table using Cloudformation

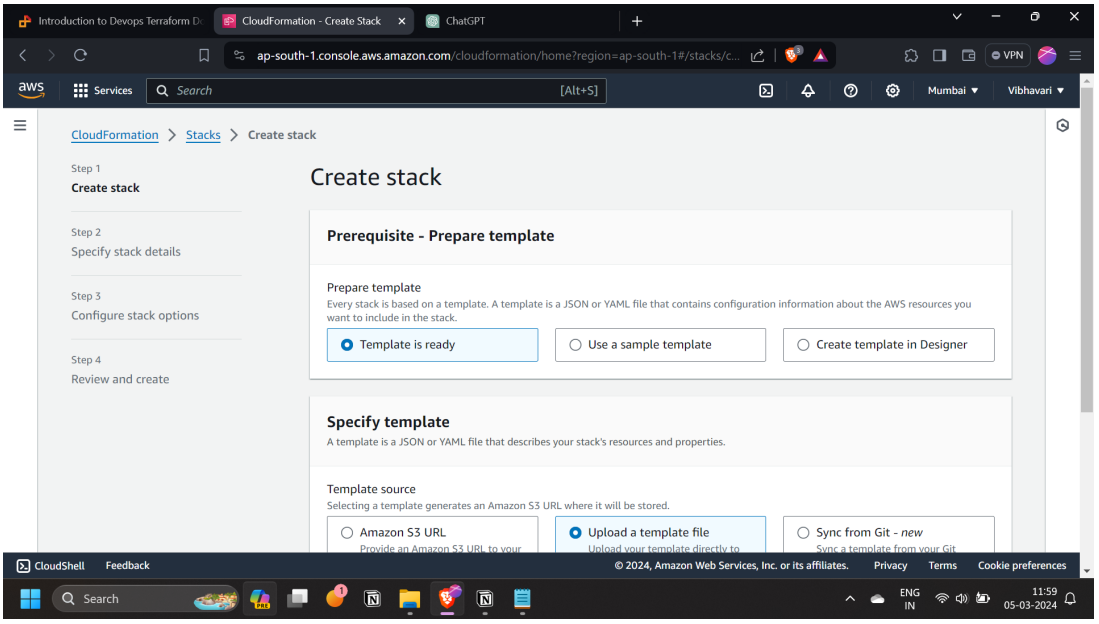
Creating a DynamoDB table using AWS CloudFormation:

1. **Authoring the CloudFormation Template**:

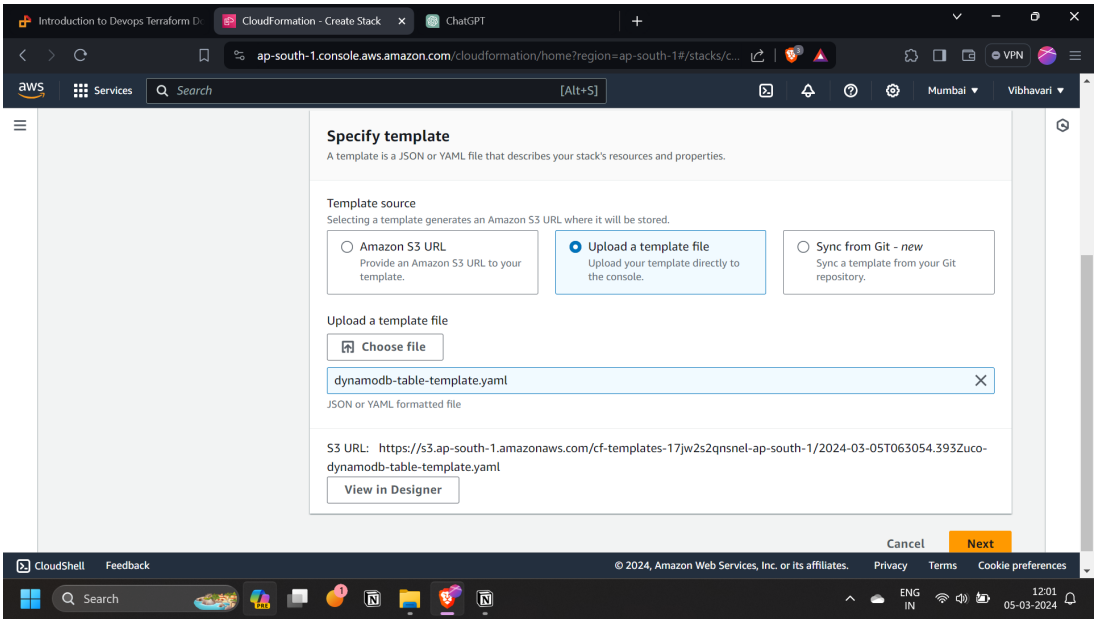
   Template

   ```
   Resources:
     ProductTable:
       Type: 'AWS::DynamoDB::Table'
       Properties:
         TableName: Product
         AttributeDefinitions:
           - AttributeName: productId
             AttributeType: S
         KeySchema:
           - AttributeName: productId
             KeyType: HASH
         ProvisionedThroughput:
           ReadCapacityUnits: 5
           WriteCapacityUnits: 5
   ```

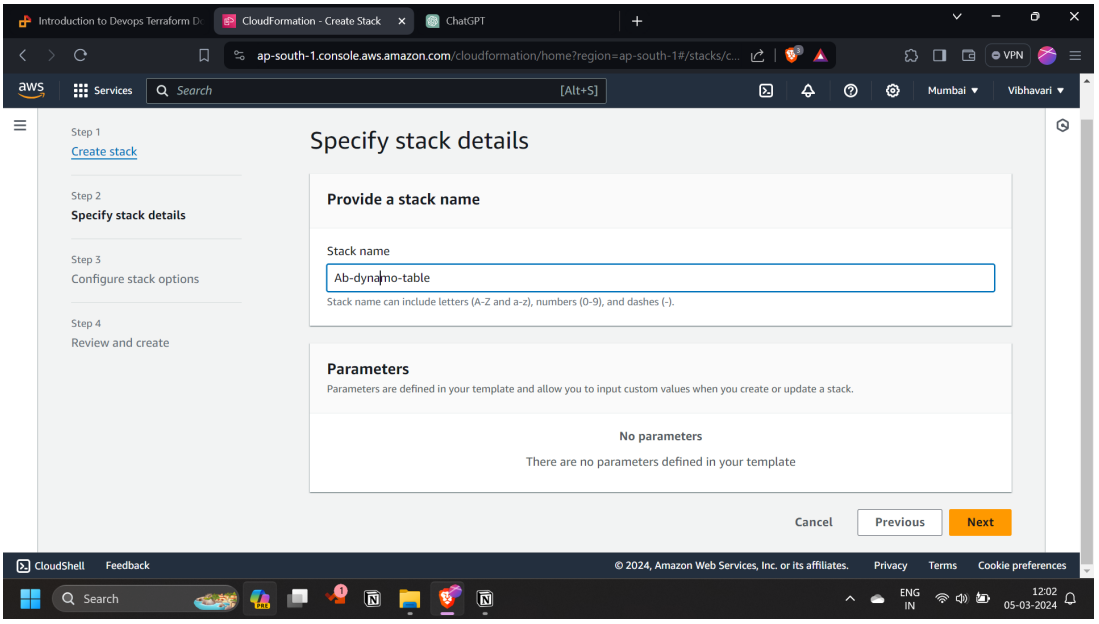
2. **Deploying the CloudFormation Stack**:
   - Click on "Create stack".



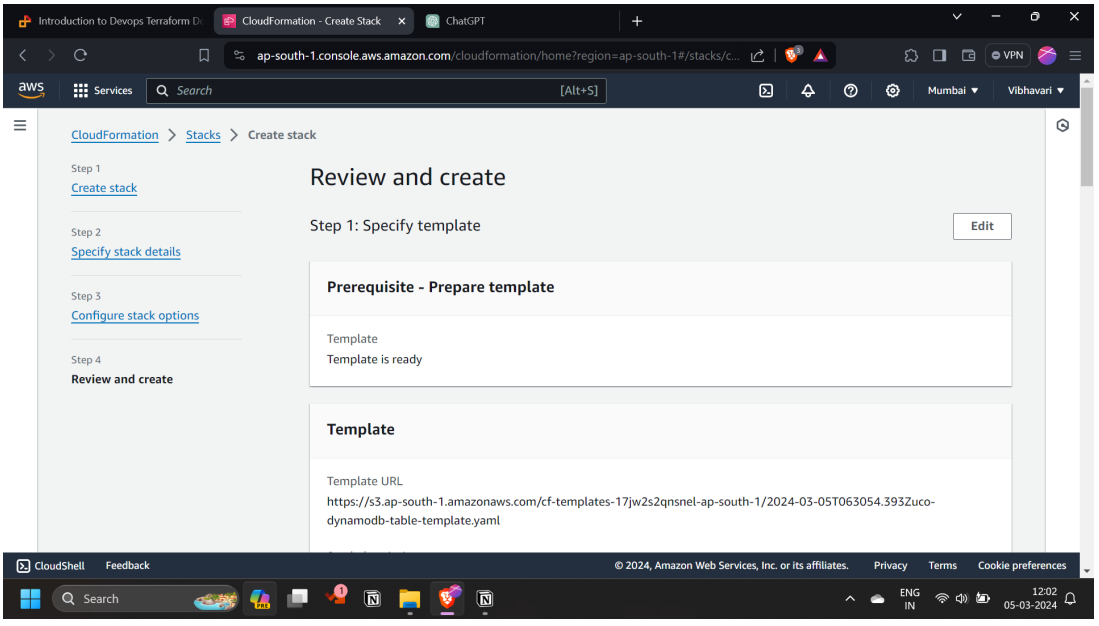
- Choose "Template is ready" and "Upload a template file", then select the template file you created.



- Enter a stack name.

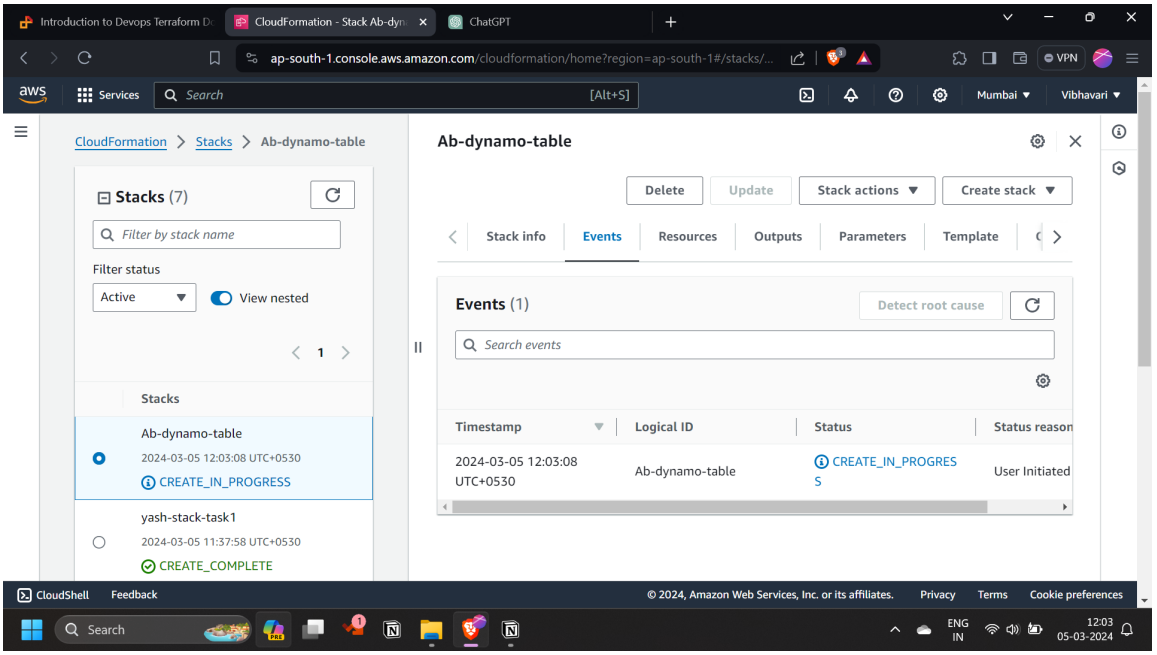


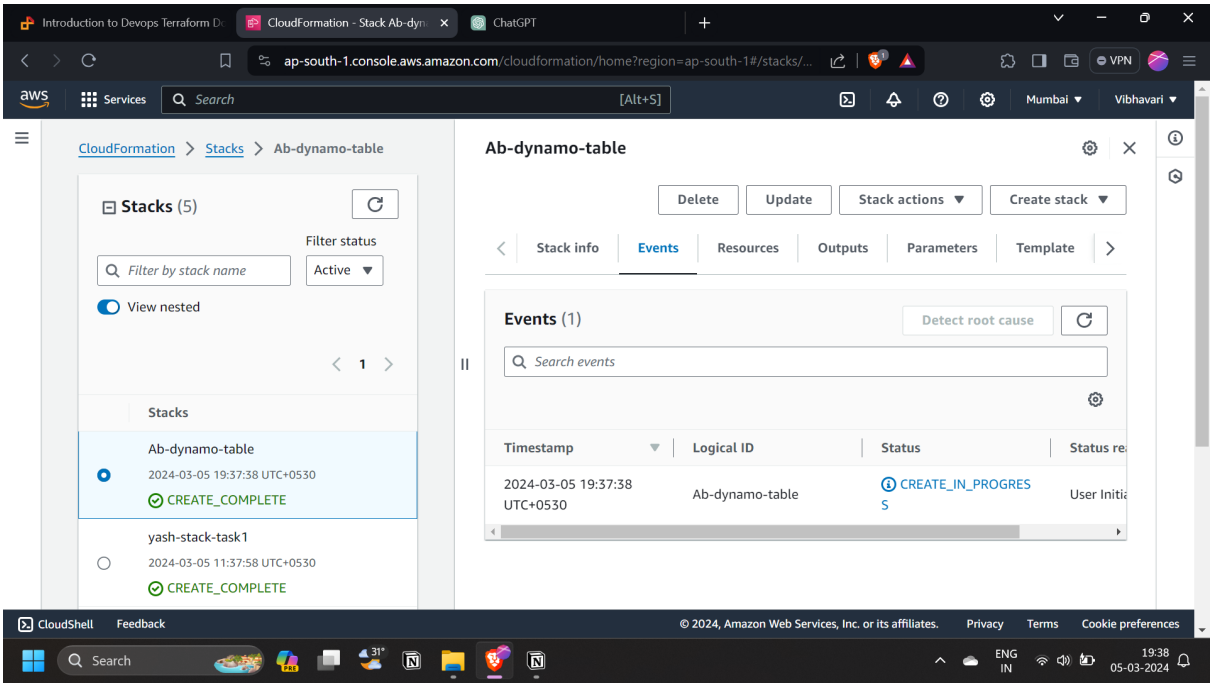
- Review the configuration and click "Create stack".



3. **Monitoring Stack Creation**:

   CloudFormation will now create the stack. You can monitor its progress in the CloudFormation dashboard. Once the stack creation is complete, your DynamoDB table will be provisioned according to the specifications in your CloudFormation template.





Verification:

After the stack creation is successful, navigate to the DynamoDB service in the AWS Management Console and verify that your table `TableName` has been created with the specified attributes and settings.

Task 7: Creating Lambda Function

Implementation of Creating Lambda Function using Cloud formation template:

Sure, here are the important steps for implementing the CloudFormation template to create a Lambda function with DynamoDB and SQS access:

1. **Prepare Your Template**: Modify the provided CloudFormation template by replacing `'YourDynamoDBTableName'` with your actual DynamoDB table name and `'YourSQSQueueURL'` with your actual SQS queue URL. Save the modified template with a `.yaml` extension.

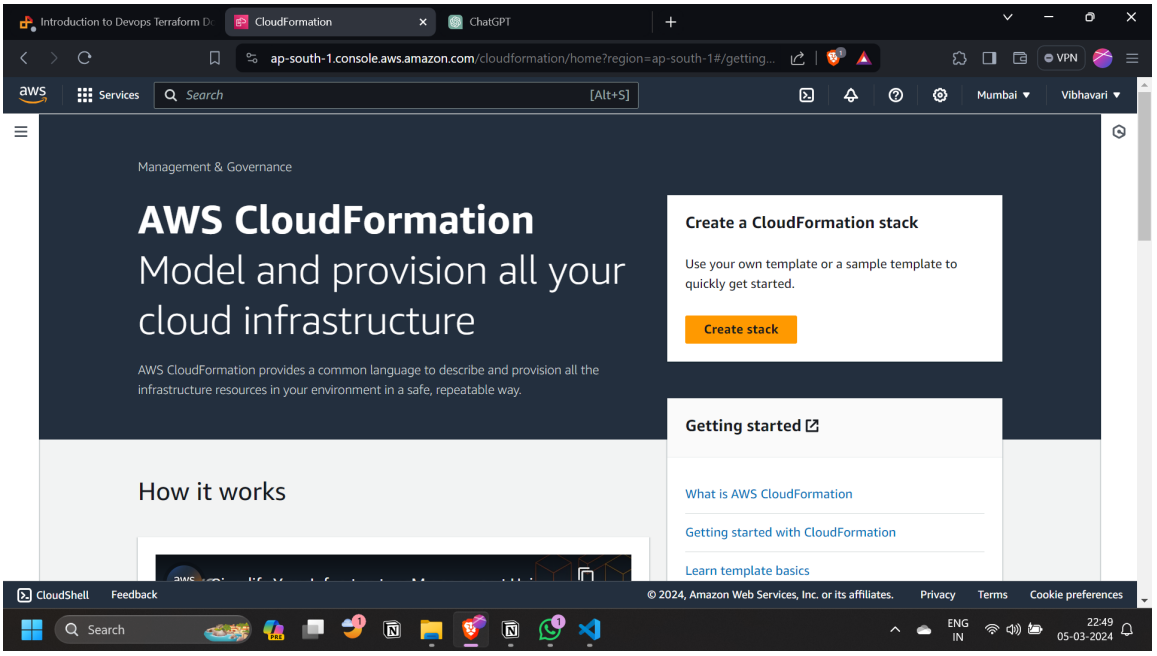
```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'CloudFormation template to create a Lambda
function with DynamoDB and SQS access'

Resources:
  LambdaExecutionRole:
    Type: 'AWS::IAM::Role'
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: 'Allow'
            Principal:
              Service: 'lambda.amazonaws.com'
            Action: 'sts:AssumeRole'
      Policies:
        - PolicyName: 'LambdaDynamoDBSQSPolicy'
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: 'Allow'
                Action:
                  - 'dynamodb:PutItem'
                Resource: '*'
              - Effect: 'Allow'
                Action:
                  - 'sqs:SendMessage'
                Resource: 'arn:aws:sqs:ap-south-1:905418
```

```
082383:ab-queue'

  LambdaFunction:
    Type: 'AWS::Lambda::Function'
    Properties:
      Handler: 'index.lambda_handler'
      Role: !GetAtt LambdaExecutionRole.Arn
      Code:
        ZipFile: |
          import boto3
          import json
          import time

          dynamodb = boto3.client('dynamodb')
          sqs = boto3.client('sqs')

          table_name = 'YourDynamoDBTableName'
          queue_url = 'YourSQSQueueURL'
          max_retries = 3

          def lambda_handler(event, context):
              try:
                  data = json.loads(event['body'])

                  response = dynamodb.put_item(
                      TableName=table_name,
                      Item={
                          'id': {'S': data['id']},
                          'payload': {'S': json.dumps(da
ta)}
                      }
                  )

                  sqs.send_message(
                      QueueUrl=queue_url,
                      MessageBody=json.dumps(data)
                  )
```

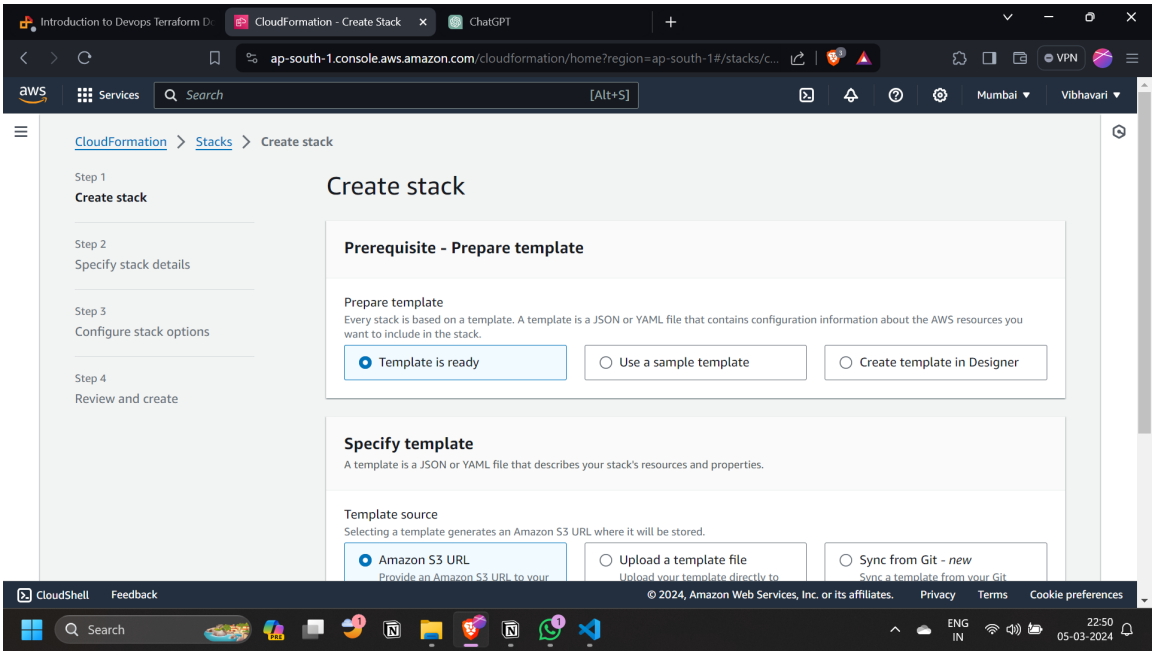
```
                    return {
                        'statusCode': 200,
                        'body': json.dumps('Data stored in
DynamoDB and sent to SQS successfully')
                    }
                except Exception as e:
                    retries = 0
                    while retries < max_retries:
                        retries += 1
                        time.sleep(retries * 2)
                        try:
                            response = lambda_handler(even
t, context)
                            return response
                        except Exception as e:
                            if retries == max_retries:
                                return {
                                    'statusCode': 500,
                                    'body': json.dumps('Fa
iled to store data in DynamoDB and send to SQS after max
imum retries')
                                }
                            else:
                                continue

      Runtime: python3.8
      Timeout: 30

  LambdaPermissionDynamoDB:
    Type: 'AWS::Lambda::Permission'
    Properties:
      Action: 'lambda:InvokeFunction'
      FunctionName: !GetAtt LambdaFunction.Arn
      Principal: 'dynamodb.amazonaws.com'

  LambdaPermissionSQS:
    Type: 'AWS::Lambda::Permission'
    Properties:
```

```
      Action: 'lambda:InvokeFunction'
      FunctionName: !GetAtt LambdaFunction.Arn
      Principal: 'sqs.amazonaws.com'

Outputs:
  LambdaFunctionARN:
    Description: 'ARN of the created Lambda function'
    Value: !GetAtt LambdaFunction.Arn
```

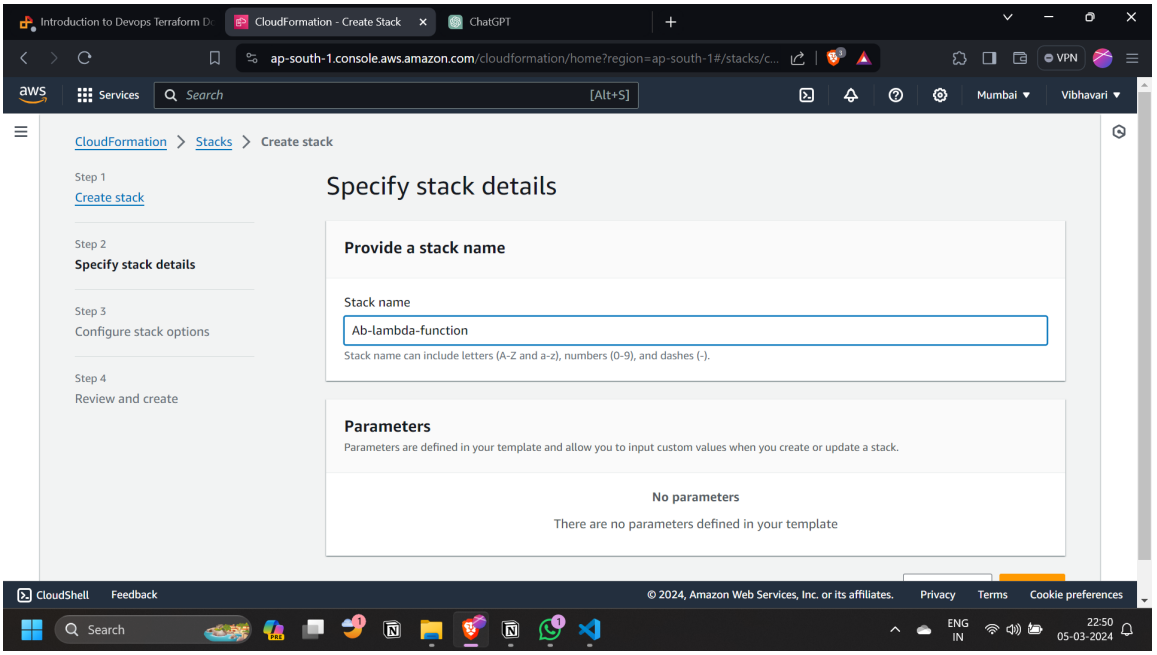
2. **Navigate to CloudFormation**: Go to the AWS Management Console and navigate to the CloudFormation service.



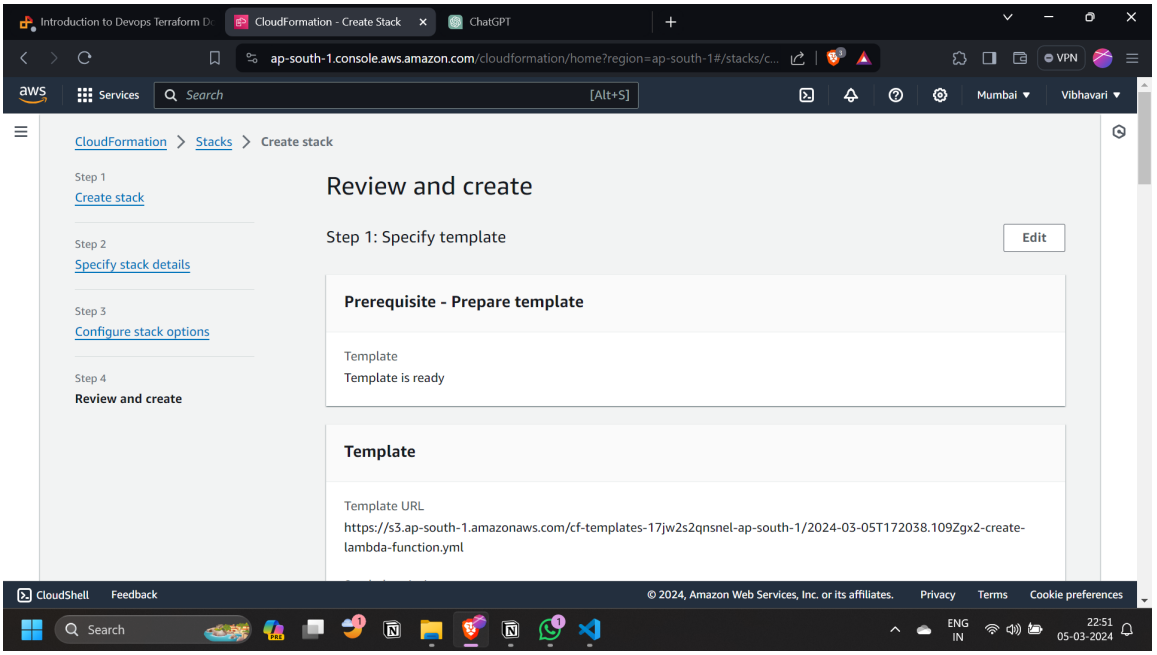
3. **Create a New Stack**: Click on "Create stack" and upload the modified CloudFormation template file.



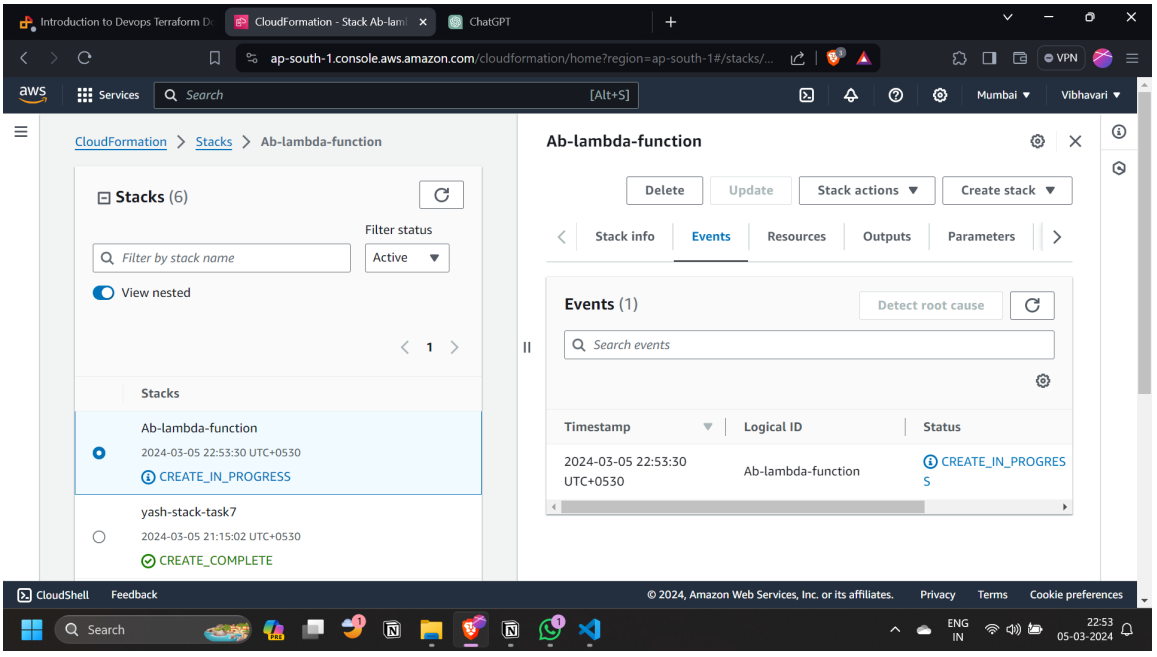
4. **Specify Stack Details**: Enter a stack name and click "Next".

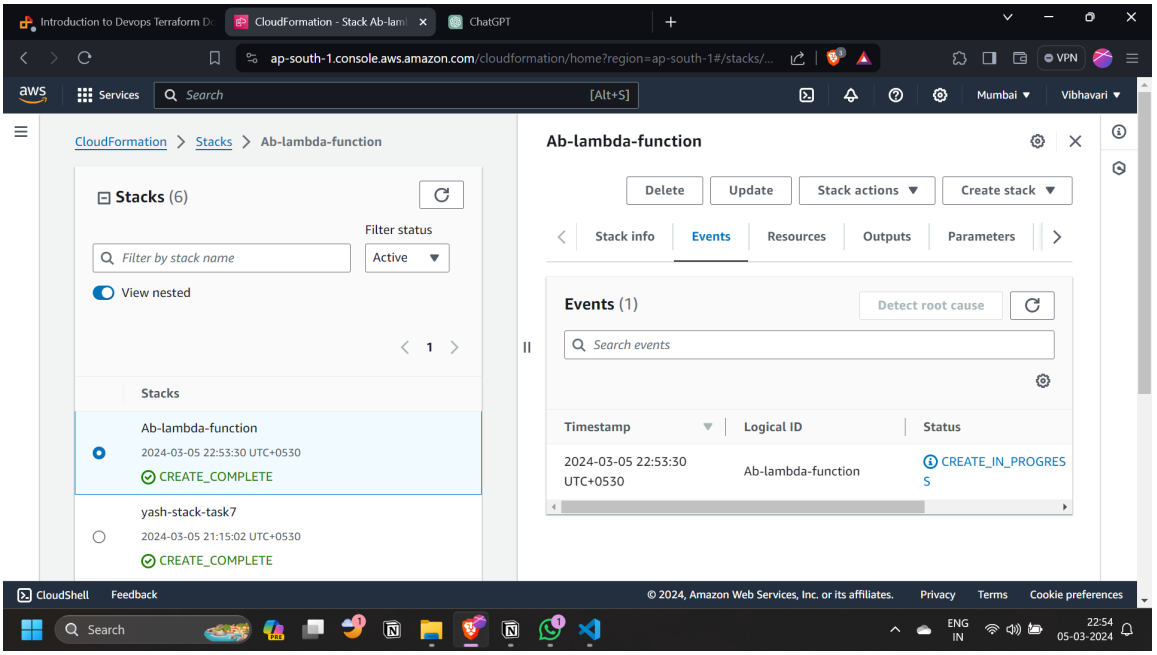


5. **Review and Create Stack**: Review the stack details and click "Create stack" to initiate the creation process.

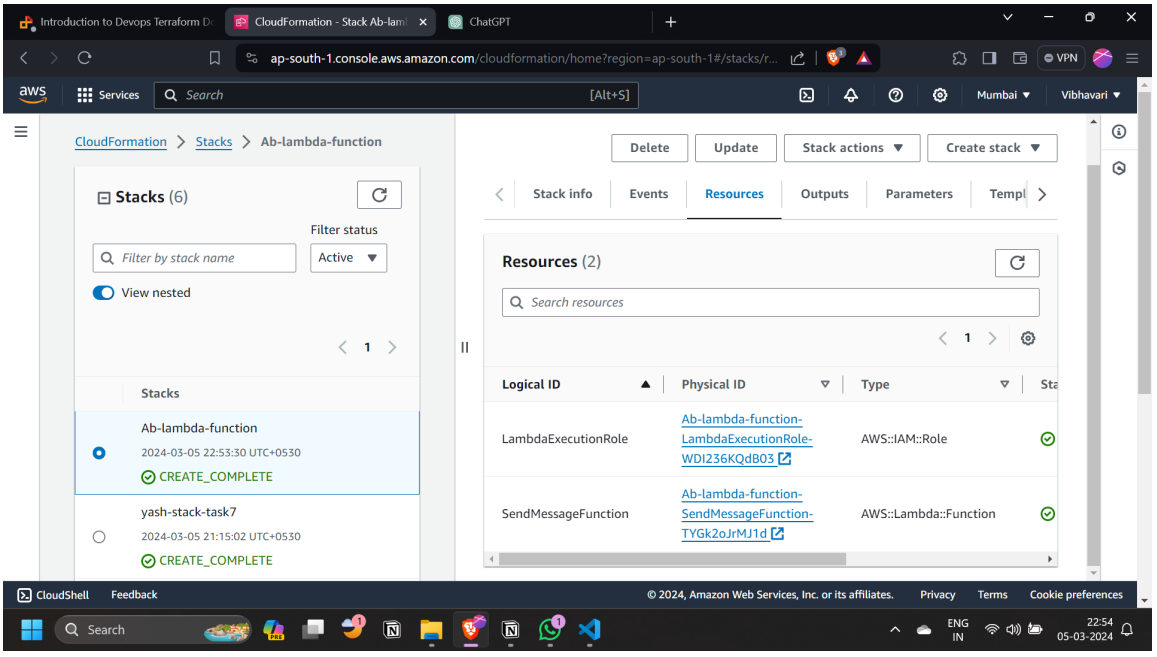


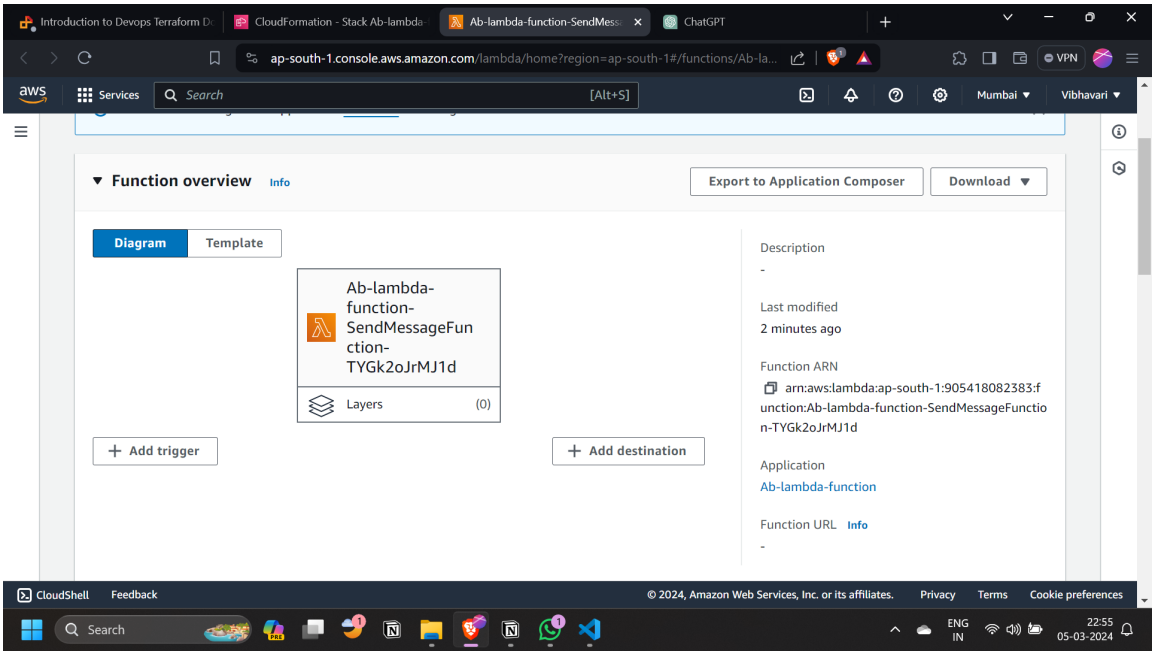
6. **Wait for Stack Creation**: Monitor the CloudFormation console until the stack status changes to "CREATE_COMPLETE".



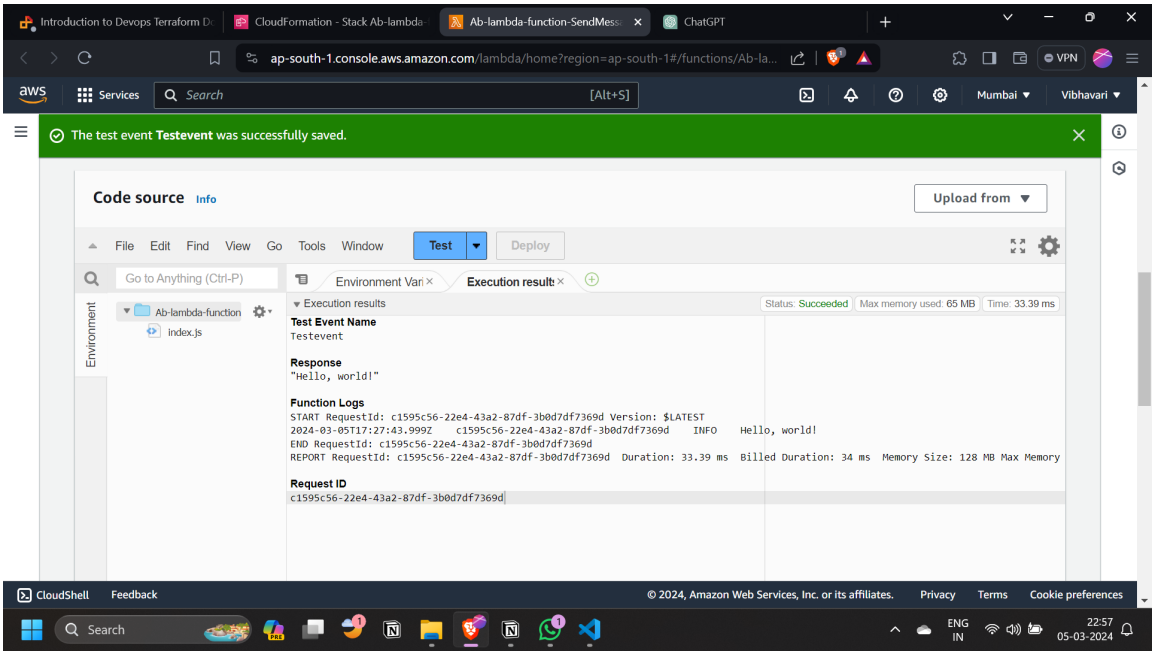


7. **Verify Lambda Function**: Confirm that the Lambda function is created by navigating to the Lambda service in the AWS Management Console.





8. **Test Lambda Function**: Test the Lambda function by configuring a test event with sample data and invoking the function manually from the Lambda console.



The Lambda function to store incoming data to DynamoDB and send it to an SQS queue using the CloudFormation template provided.

Implementation of Creating Lambda Function using Cloud formation template:

Sure, here are the important steps for implementing the CloudFormation template to create a Lambda function with DynamoDB and SQS access:

1. **Prepare Your Template**: Modify the provided CloudFormation template by replacing `'YourDynamoDBTableName'` with your actual DynamoDB table name and `'YourSQSQueueURL'` with your actual SQS queue URL. Save the modified template with a `.yaml` extension.

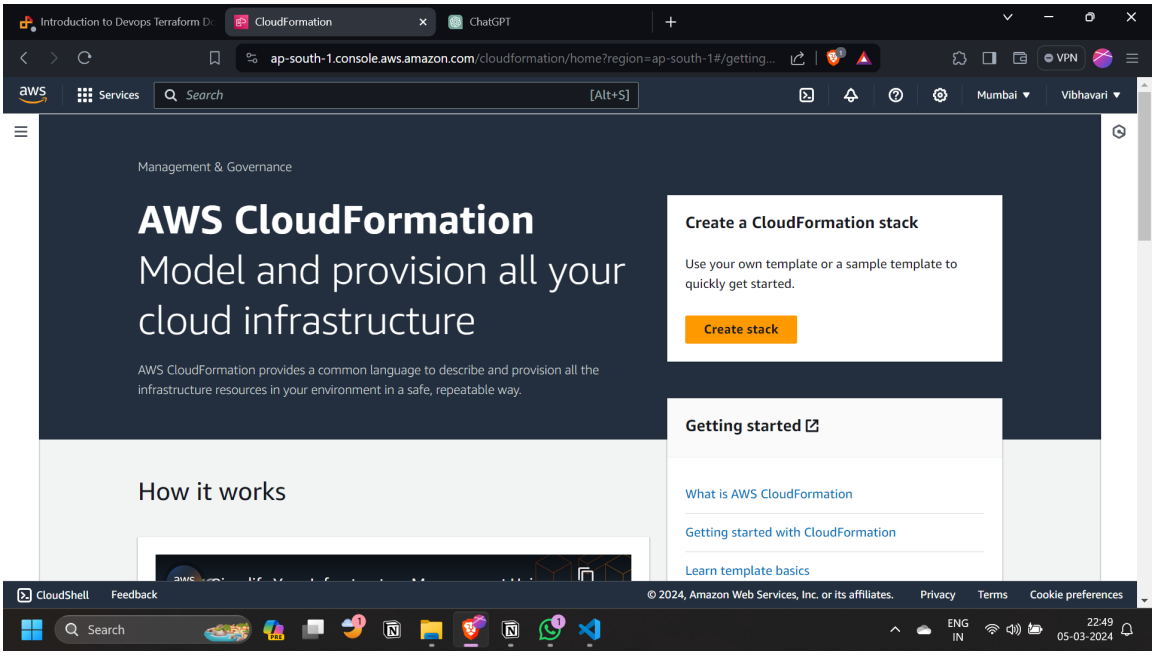
```
AWSTemplateFormatVersion: '2010-09-09'
Description: 'CloudFormation template to create a Lambda
function with DynamoDB and SQS access'

Resources:
  LambdaExecutionRole:
    Type: 'AWS::IAM::Role'
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: 'Allow'
            Principal:
              Service: 'lambda.amazonaws.com'
            Action: 'sts:AssumeRole'
      Policies:
        - PolicyName: 'LambdaDynamoDBSQSPolicy'
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: 'Allow'
                Action:
                  - 'dynamodb:PutItem'
                Resource: '*'
              - Effect: 'Allow'
                Action:
                  - 'sqs:SendMessage'
                Resource: 'arn:aws:sqs:ap-south-1:905418
```

```
082383:ab-queue'

  LambdaFunction:
    Type: 'AWS::Lambda::Function'
    Properties:
      Handler: 'index.lambda_handler'
      Role: !GetAtt LambdaExecutionRole.Arn
      Code:
        ZipFile: |
          import boto3
          import json
          import time

          dynamodb = boto3.client('dynamodb')
          sqs = boto3.client('sqs')

          table_name = 'YourDynamoDBTableName'
          queue_url = 'YourSQSQueueURL'
          max_retries = 3

          def lambda_handler(event, context):
              try:
                  data = json.loads(event['body'])

                  response = dynamodb.put_item(
                      TableName=table_name,
                      Item={
                          'id': {'S': data['id']},
                          'payload': {'S': json.dumps(da
ta)}
                      }
                  )

                  sqs.send_message(
                      QueueUrl=queue_url,
                      MessageBody=json.dumps(data)
                  )
```

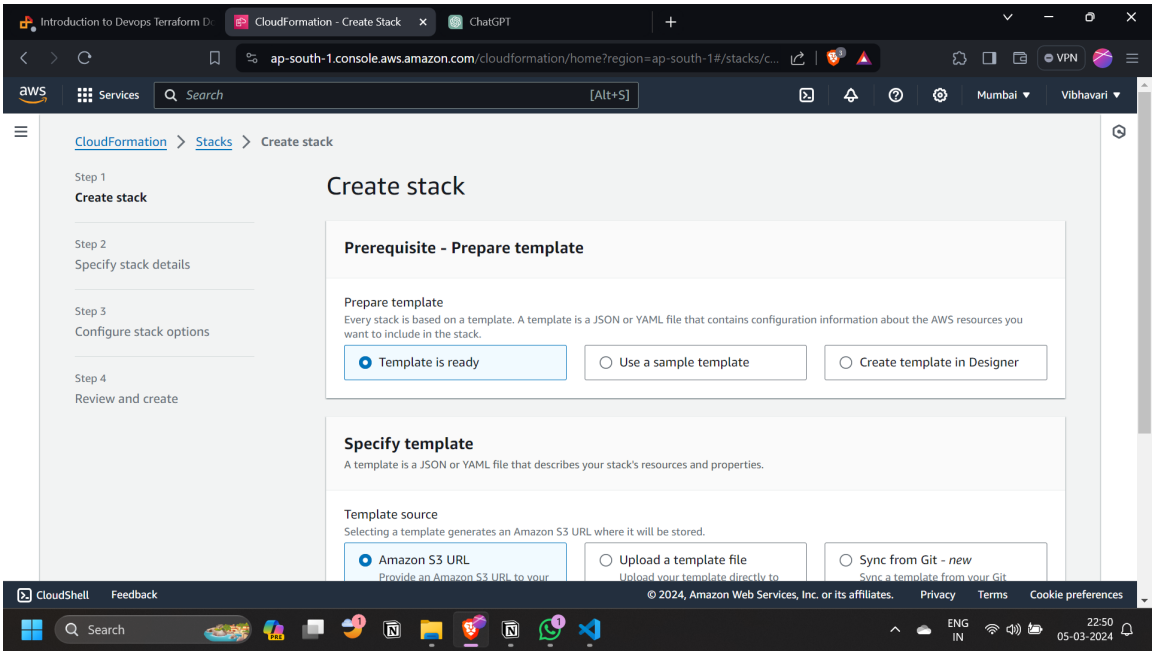
```
                    return {
                        'statusCode': 200,
                        'body': json.dumps('Data stored in
DynamoDB and sent to SQS successfully')
                    }
                except Exception as e:
                    retries = 0
                    while retries < max_retries:
                        retries += 1
                        time.sleep(retries * 2)
                        try:
                            response = lambda_handler(even
t, context)
                            return response
                        except Exception as e:
                            if retries == max_retries:
                                return {
                                    'statusCode': 500,
                                    'body': json.dumps('Fa
iled to store data in DynamoDB and send to SQS after max
imum retries')
                                }
                            else:
                                continue

      Runtime: python3.8
      Timeout: 30

  LambdaPermissionDynamoDB:
    Type: 'AWS::Lambda::Permission'
    Properties:
      Action: 'lambda:InvokeFunction'
      FunctionName: !GetAtt LambdaFunction.Arn
      Principal: 'dynamodb.amazonaws.com'

  LambdaPermissionSQS:
    Type: 'AWS::Lambda::Permission'
    Properties:
```

```
      Action: 'lambda:InvokeFunction'
      FunctionName: !GetAtt LambdaFunction.Arn
      Principal: 'sqs.amazonaws.com'

Outputs:
  LambdaFunctionARN:
    Description: 'ARN of the created Lambda function'
    Value: !GetAtt LambdaFunction.Arn
```

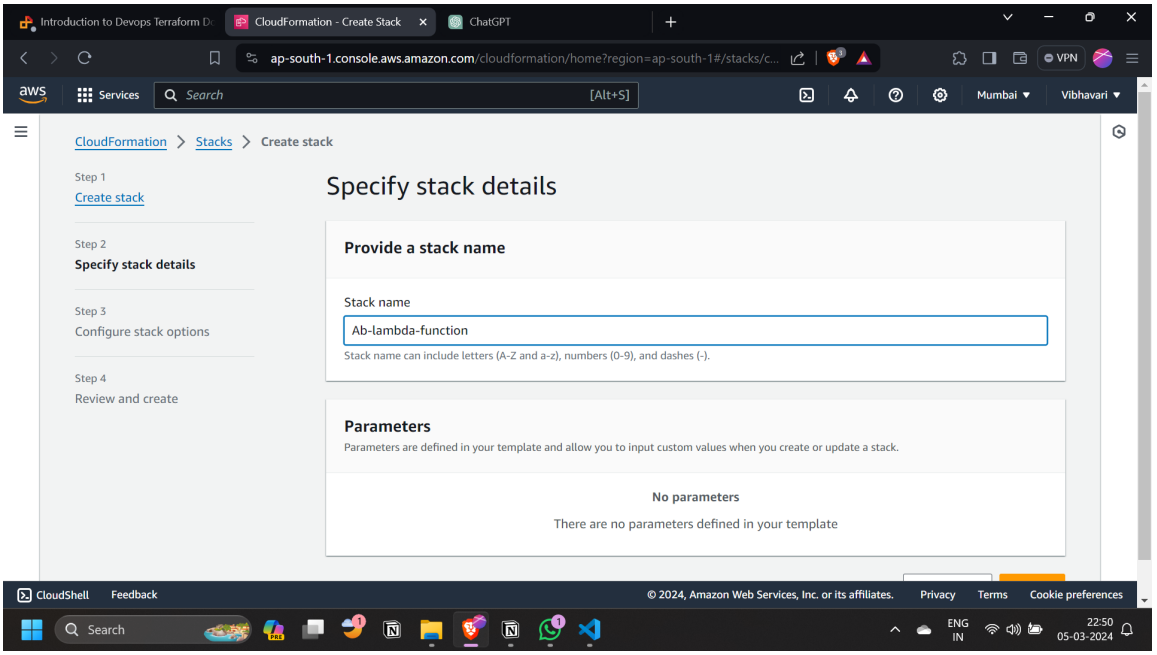
2. **Navigate to CloudFormation**: Go to the AWS Management Console and navigate to the CloudFormation service.



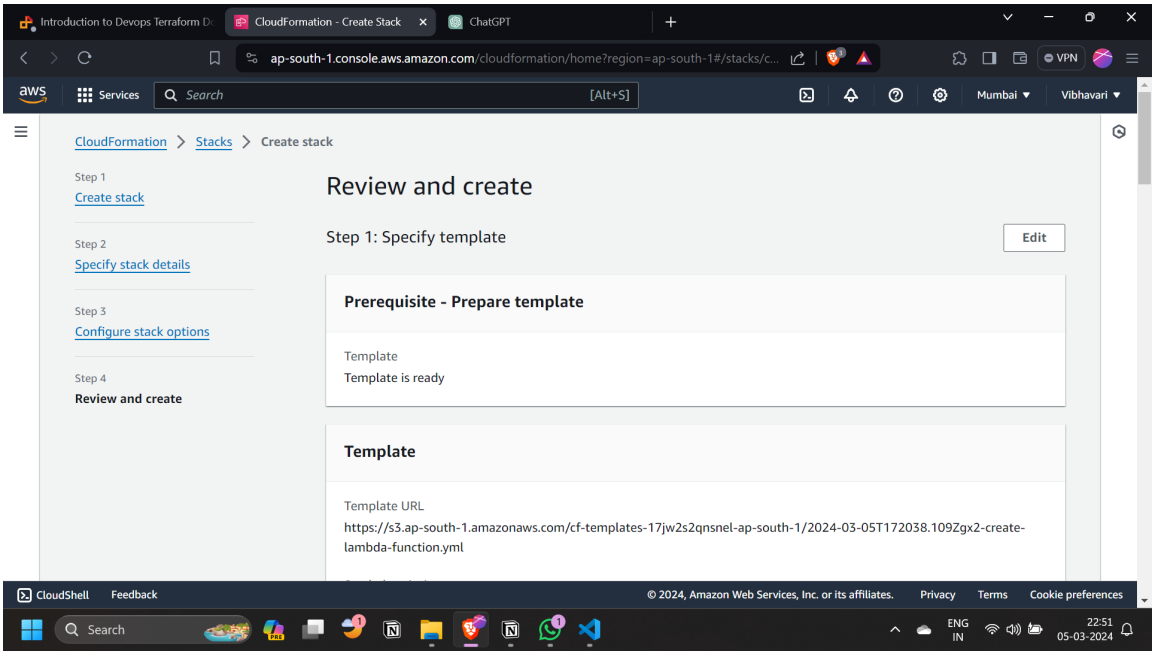
3. **Create a New Stack**: Click on "Create stack" and upload the modified CloudFormation template file.



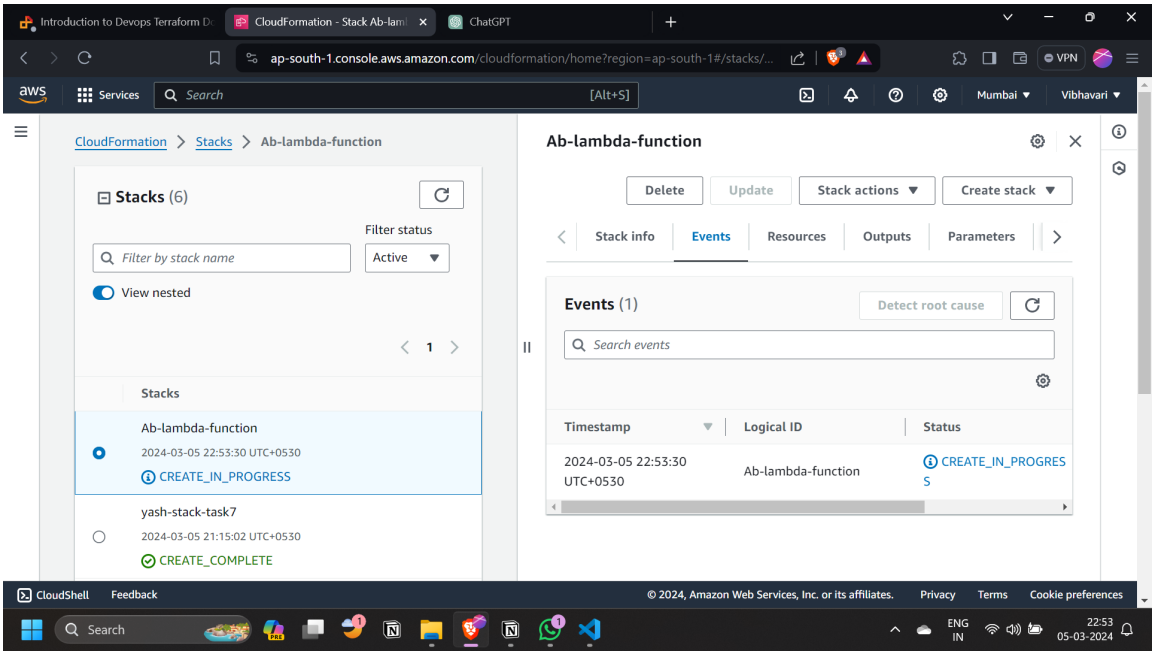
4. **Specify Stack Details**: Enter a stack name and click "Next".

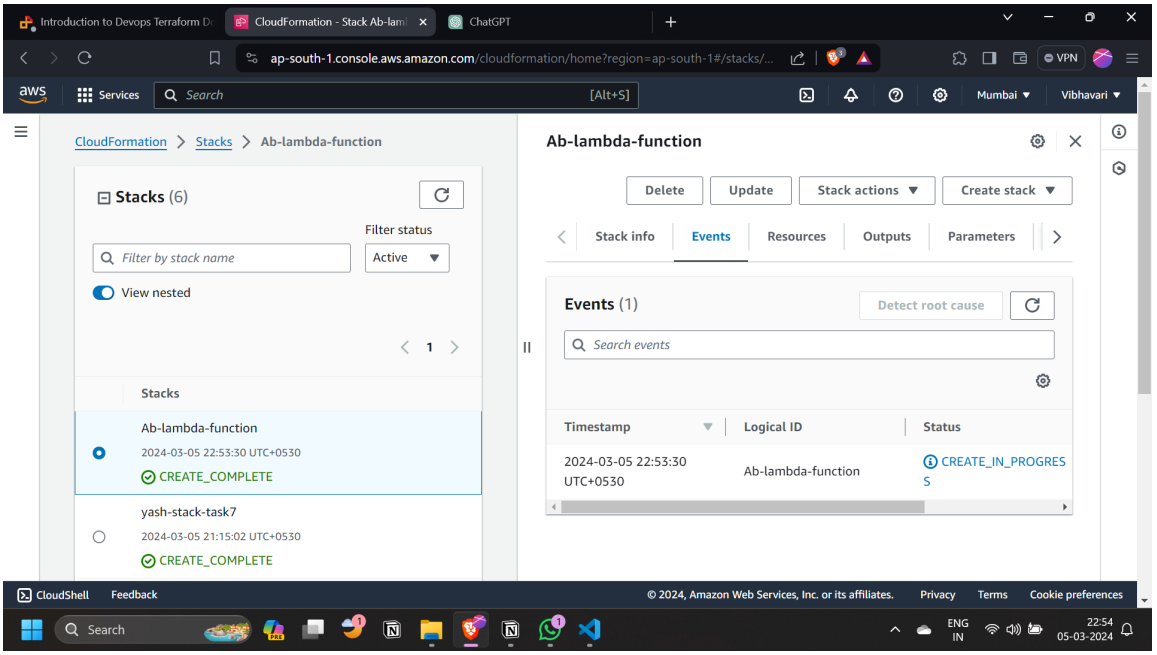


5. **Review and Create Stack**: Review the stack details and click "Create stack" to initiate the creation process.

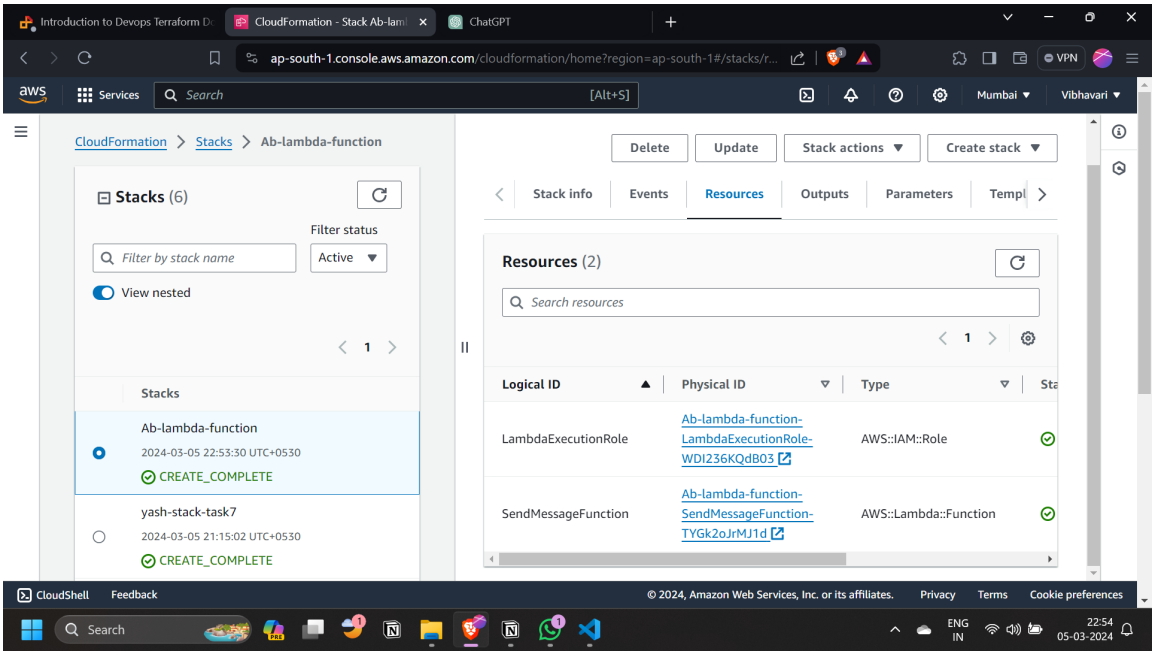


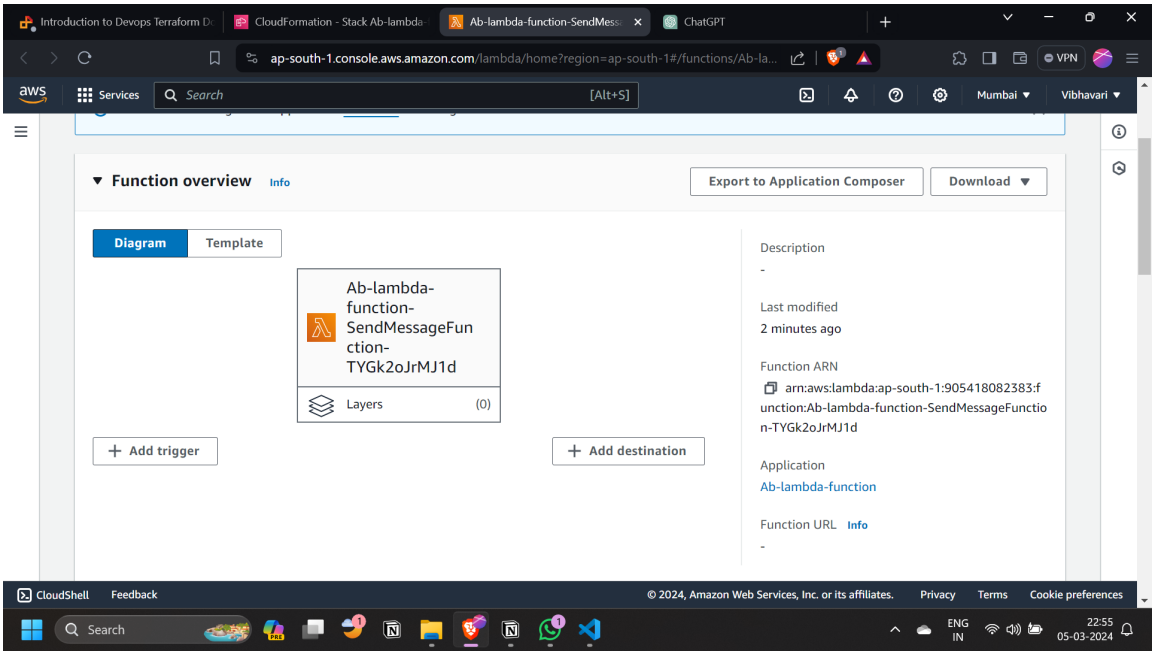
6. **Wait for Stack Creation**: Monitor the CloudFormation console until the stack status changes to "CREATE_COMPLETE".



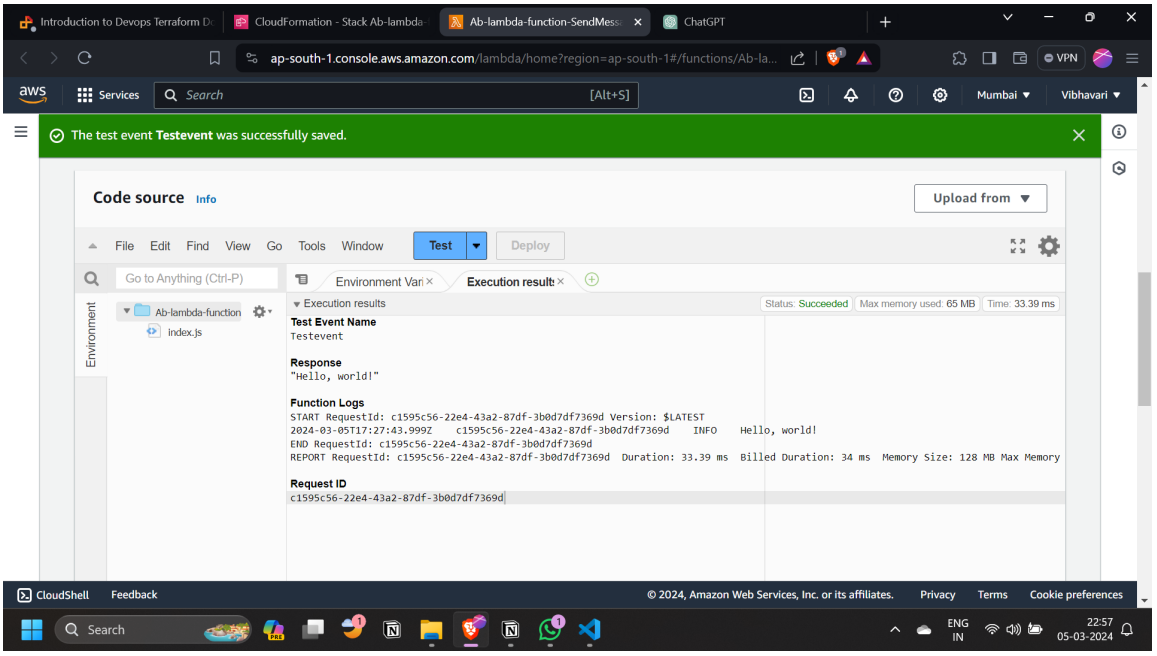


7. **Verify Lambda Function**: Confirm that the Lambda function is created by navigating to the Lambda service in the AWS Management Console.





8. **Test Lambda Function**: Test the Lambda function by configuring a test event with sample data and invoking the function manually from the Lambda console.



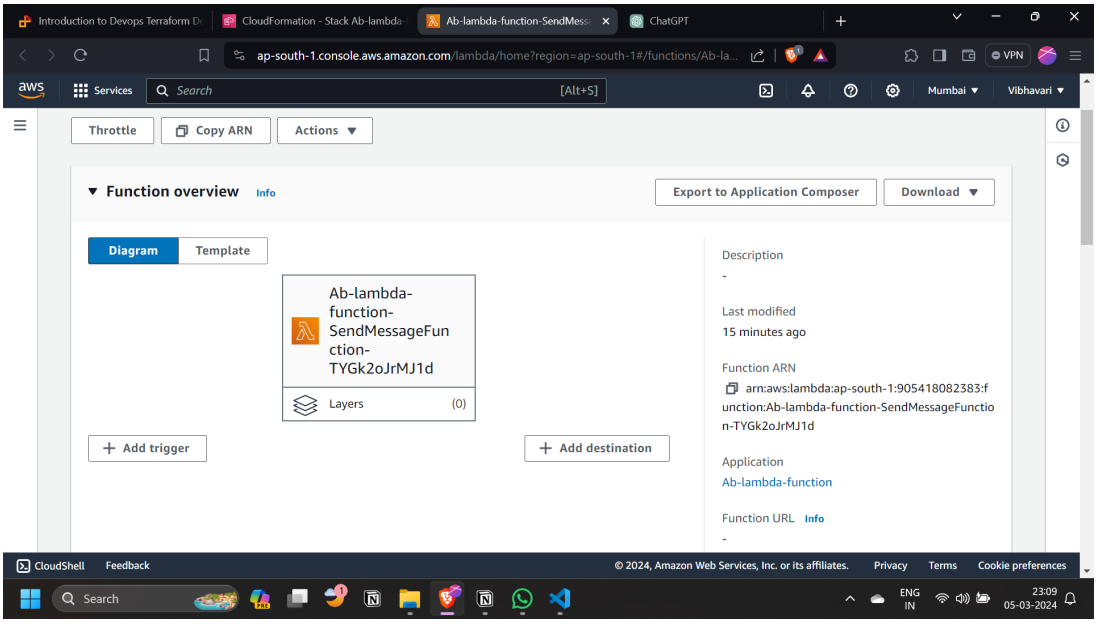
The Lambda function to store incoming data to DynamoDB and send it to an SQS queue using the CloudFormation template provided.

Task 8: Attach Lambda to SQS queue

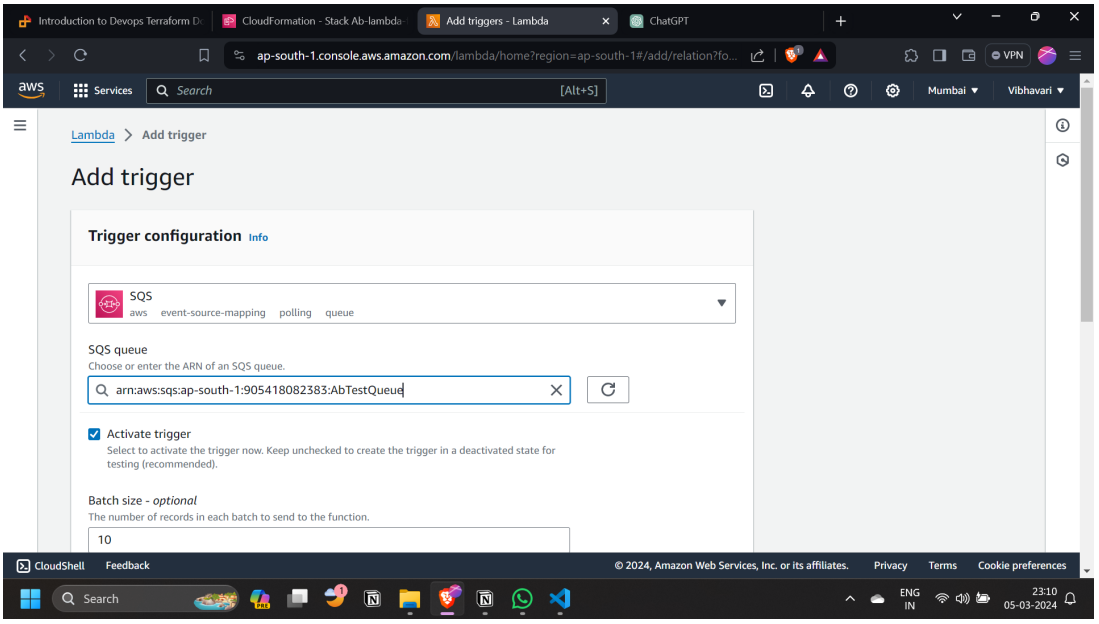
To attach a Lambda function to an SQS queue, you can use an SQS trigger configuration in the Lambda function settings. This configuration ensures that the Lambda function is invoked whenever a message is available in the SQS queue.

Here's how you can achieve SQS integration with Lambda and store data to a DynamoDB table when data comes to the SQS queue:

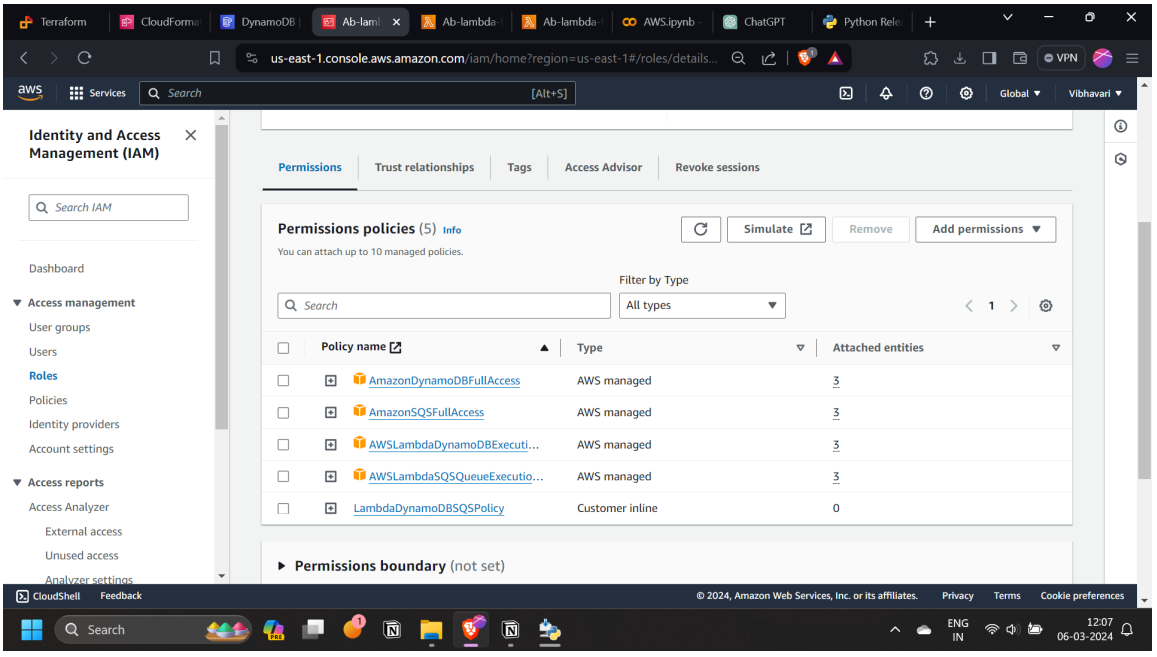
1. **Lambda Function**:
   First, ensure that you have a Lambda function created. If not, create a Lambda function that will process messages from the SQS queue and store data to the DynamoDB table as per your requirements.
2. **SQS Queue**:
   Make sure you have an SQS queue created where data will be sent. If not, create an SQS queue in the AWS Management Console.
3. **Configure Lambda-SQS Integration**:
   Go to the Lambda service in the AWS Management Console. Find the Lambda function you want to attach to the SQS queue. Then, follow these steps:
   - Open the Lambda function configuration.



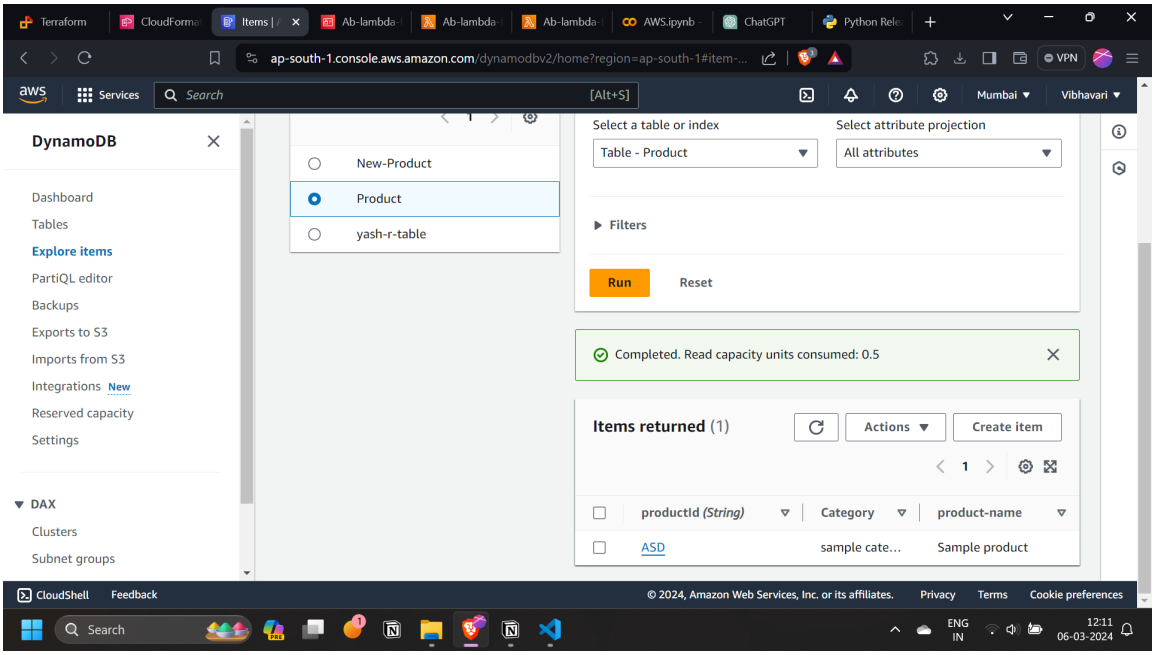
   - Click on "Add trigger".



4. **Ensure Lambda Permissions**:
   Ensure that the Lambda function's execution role has the necessary permissions to receive messages from the SQS queue. You might need to add the
   `SQS:ReceiveMessage` permission to the Lambda execution role



5. **Testing**:
   Test the integration by sending messages to the SQS queue. Monitor the Lambda function's execution to verify that it processes the messages and stores the data in the DynamoDB table correctly.



Python code to send data to SQS

```
import boto3
import json

# Create an SQS client
region = 'ap-south-1'
# Replace 'your_access_key_id' and 'your_secret_access_key'
with your AWS credentials
aws_access_key_id = 'AKIAYS2NU4DZW5NYCCFS'
aws_secret_access_key = 'ZUwKcoxwbEtM2yqXVQhXmp4ce0muVvztGv
NoLLri'

# URL of your SQS queue
queue_url = '<https://sqs.ap-south-1.amazonaws.com/90541808
2383/ab-queue>'

sqs = boto3.client('sqs', region_name=region, aws_access_ke
y_id=aws_access_key_id, aws_secret_access_key=aws_secret_ac
cess_key)

# Data to send
data = {
    'ProductID': '123467',
```

```
    'ProductName': 'Sample Product',
    'Quantity': 200,
    'Price': 50,
    'Discount': 5,
    'Category': 'Sample Category',
    'RemainingStock': 100
}

# Send data to SQS
response = sqs.send_message(
    QueueUrl=queue_url,
    MessageBody=json.dumps(data)
)

print("Message sent to SQS:", response['MessageId'])
```

By following these steps, you can attach a Lambda function to an SQS queue and ensure that data sent to the SQS queue is processed and stored in the DynamoDB table by the Lambda function.